

Prix: 740 F par abonnement.

Sommaire de la revue :



Présentation des disquettes ToolBox Mag Numéro 7

Deux disquettes, il fallait bien ça. Voici leurs catalogues:

Disquette 7A :

/ANIM.07: Ils s'y sont tous mis, pour ce numéro-anniversaire. Leur nom défilera si vous bootez la disquette. Festival Graphique/Son…

/ANNONCES: Les petites annonces gratuites.

/ASM.TML: Nous cherchions tous comment coller de l'assembleur dans TML Pascal: Stéphane Hadinger l'avait fait depuis longtemps pour Fontasm! Voyez pages 20-21.

/DERNIERE.HEURE: lisez le fichier *Lisez.Moi*.

/DESTEL.GOODIES1/FONTES: Deux fontes-écran de Jean Destelle. Elles sont destinées à l'affichage en mode 640.

/DESTEL.GOODIES1/NDA: Un accessoire de conversion Décimal/Hexadécimal de Jean Destelle.

/GATE.IMAGES: Deux écrans de Gate pour illustrer nos articles sur ce jeu récent. Voyez pages 22-23.

/POSTSCRIPT: Les deux programmes PostScript de Jean-Yves Bourdin dont vous voyez le résultat pages 13 et 15. Voyez l'article pages 11-16.

/RAMTOOLS.CDEV: Philippe Manet prend spectaculairement la suite de F. Uhrich à la "chaire de C" de ToolBox-Mag. Voyez pages 4 à 10.

/RESDOCTOR.1.05: Pour fêter son entrée dans notre Conseil de Rédaction, où il remplace F. Uhrich, Jean Destelle nous offre une nouvelle version de ResDoctor, analysant les ressources-version décrites par B. Fournier dans notre numéro 5.

/SOMMAIRE.1.6: En base de données AppleWorks-GS pour faciliter vos recherches, le contenu des pages «+1» à «+4».

PRODOS est toujours un faux: ne le lancez pas sous Finder!

Disquette 7B :

/DESTEL.GOODIES2: Regardez donc ce que fait l'application de Jean Destelle. Pas mal, non?

/FNTS.TOURNEREAU: De magnifiques fontes-écran et ImageWriter de M. Tournereau. Après les avoir installées, chargez et imprimez les fichiers AppleWorks-GS du disque.

/HYPERMEDIA: Une pile HyperStudio de «clip-art» de Roger Bégin. Avec le couper-coller, utilisez ces images à votre façon.

/PUZZLE.IMAGES: Des images très belles, mais plutôt vicieuses, de Robert Santelli pour le jeu de puzzle de notre n° 2.

/RESCONV: L'application de notre "ressourcier" national, Bernard Fournier. Voyez pages 32 à 35.

ToolBox-Mag

Directeur de la Publication
Eric Weyland

Rédacteur en Chef
Jean-Yves Bourdin

Mise en pages
Dominique Digoy

Secrétariat
Janine Loiseleux

Conseil de Rédaction
Jean-Pierre Charpentier
Dominique Delay
Patrick Desnoues
Jean Destelle
Bernard Fournier
Olivier Goguel
Stéphane Hadinger
François Hermellin
Hubert Loiseleux
Claude Pélisson
Jean-Luc Schmitt

Rédaction, Siège Social
6, rue Henri-Barbusse
95100 Argenteuil - France

Impression
IRG - Argenteuil

Les auteurs du n° 7
Olivier Bailly-Maître
Roger Bégin
Nicolas Bergeret
Jean-Yves Bourdin
Laurent Bourdin
Dominique Delay
Jean Destelle
Steven W. Disbrow
Bernard Fournier
Michaël Guitton
Olivier Goguel
Stéphane Hadinger
Yvan Kœnig
Philippe Manet
Sébastien Mousseron
Robert Santelli
Jean-Louis Torre
M. Tournereau
Jean-Michel Vallat
Eric Weyland

Avertissements légaux

ToolBox-Mag est un magazine qui prend l'Apple II GS au sérieux. De ce fait même, il est indépendant d'Apple Computer et d'Apple Computer France, auxquels il n'est, grâce au ciel, pas rattaché.

ToolBox-Mag est disponible uniquement par abonnement et par correspondance. Le magazine est composé inséparablement d'une ou plusieurs disquettes magnétiques pour ordinateur Apple // GS et d'une revue sur papier.

Magazine indépendant, ToolBox-Mag permet à ses auteurs de s'exprimer sous leur propre responsabilité. Leurs propos n'engagent ni ToolBox-Mag ni la Société Toolbox.

ToolBox-Mag ne peut pas offrir une garantie supérieure à celle qu'offre le chapitre "Limitations de Garantie et de Responsabilité" des documentations officielles Apple, soit quasiment rien.
Ecrit par des amateurs du GS, il s'engage néanmoins à publier des programmes ne contenant pas plus de bugs que GS Write, et à maintenir un standard de compétence au moins égal au niveau moyen des concessionnaires Apple français en matière d'Apple II GS...

L'Editorial du Directeur :

Premier anniversaire

Oui, un an déjà: une fois le champagne éclusé, ce joyeux anniversaire a été l'occasion pour nous de jeter un coup d'œil sur notre collection. Que de changements en un an! Le premier numéro comportait six auteurs, et le numéro 7 en compte... vingt! La première disquette affichait 'Unable to Load Prodos' au boot, lancez donc les deux disquettes de ce numéro pour voir la différence. ToolBox-Mag cherchait un peu sa physionomie lors des premiers numéros, il l'a aujourd'hui trouvée.
La communauté Apple II GS est vivante et active dans notre pays. Il existe non seulement des individus, non seulement des groupes, mais bel et bien un **marché** de l'informatique sans attaché-case, de l'ordinateur pour le plaisir.

Ce coup d'œil sur le passé nous donne aussi l'occasion d'un petit appel au peuple: ToolBox-Mag est **votre** magazine, le magazine de tous ceux qui aiment leur GS. **Tous**, vous pouvez y contribuer: regardez les contributions de Robert Santelli, M. Tournereau, Roger Bégin à ce numéro. Elles sont intéressantes, amusantes, instructives, et il n'y a pas besoin de connaître les trois techniques de codage de l'algorithme de Van Halen/Alzheimer pour les faire!
Certains d'entre vous souhaitent que la partie pour «utilisateurs-pas-calés-en-C++» se renforce dans ToolBox-Mag: nous sommes du nombre! Vous utilisez le tableur d'AppleWorks-GS assidûment? Ecrivez-nous! Vous avez fait des triplettes de fontes qui tournent pour la LQ? Envoyez-les nous! Vous faîtes des piles HyperCard, HyperStudio: nous sommes preneurs!
Savez-vous quel est le type d'article que nous avons le plus de mal à obtenir pour ToolBox-Mag? Les revues de logiciels! Etonnant, quand même: il n'y a pas que des programmeurs parmi nos lecteurs, que Diable!
Nous vous proposons donc de lancer une série d'articles «Les Classiques du GS». Vous, car c'est à **vous** de les écrire, y expliquerez comment vous utilisez Prosel-16, Wings, AppleWorks-GS, comment vous gagnez aux jeux de CinemaWare ou à Rastan, etc, etc. Rendez-vous compte: en un an, nous n'avons toujours pas parlé de tous les logiciels ci-dessus, ni des autres programmes majeurs du GS! C'est vrai, il y a la pression du tout nouveau, du qui-vient-de-sortir. Mais justement, sur GS, cette pression est moins étouffante qu'ailleurs: profitons-en.

L'informatique sans cravate, c'est aussi **l'informatique sans complexes**: comparé aux autres machines, le GS est amusant et, dans le fond, plutôt facile. Pour cette seconde année, nous aimerions que ToolBox-Mag devienne un peu plus **votre** magazine. Venez en discuter à l'Apple-Expo, au stand Toolbox, en $1B25 (bonne idée, ça, la numérotation des stands en hexa). A bientôt de vous voir... et de vous lire!

ERIC WEYLAND

RamTools



Philippe Manet

Dans le numéro 4 de ToolBox-Mag était présentée l'Init *ToolsToRam* de Patrick Desnoues chargeant les outils les plus fréquemment utilisés en ram au moment du démarrage de l'Apple IIGS, évitant ainsi les changements de disquette pour ceux qui n'ont pas encore de disque dur, et accélérant le démarrage des applications pour tout le monde.
A la fin de l'article, Jean-Yves Bourdin demandait un programme de configuration de l'Init pour choisir les outils à charger. Il m'a semblé que cela correspondait exactement à la définition d'un CDev, et que c'était un bon moyen de se lancer dans l'écriture de cette bête: le résultat se trouve sur la disquette de ce numéro.

Qu'est ce qu'un CDev ?

Un CDev *(Control Panel Device)* est un module du tableau de bord graphique apparu avec le système 5.0, et permettant de configurer un ou plusieurs des paramètres de l'Apple IIGS. C'est un fichier de type $C7 situé dans le dossier */SYSTEM/CDEVS qui présente la particularité de ne contenir que des ressources. L'accessoire "Tableau de Bord" présente la liste des CDevs présents dans ce dossier dans la partie gauche de sa fenêtre sous forme d'icônes et laisse la moitié droite disponible au CDev actuellement sélectionné pour afficher ses propres informations.
Nous verrons en détail dans la deuxième partie de l'article quelles sont les ressources présentes dans un CDev, du point de vue du programmeur. Pour l'instant, voyons comment installer et utiliser le CDev ***RamTools.***

Installation et utilisation de *RamTools*

L'installation est on ne peut plus simple: il vous suffit de transférer le fichier ***RamTools*** du dossier */RamTools.CDev* de votre disquette ToolBox Mag dans le dossier */SYSTEM/CDevs* de votre disque de démarrage avec un programme sachant copier les fichiers avec ressources (le Finder, ProSel-16 ou tout autre programme suffisamment récent). C'est tout! Contrairement à ce que vous pourrez lire ailleurs, il n'est pas nécessaire de rebooter le GS pour utiliser un CDev qui vient d'être installé.
En revanche, si vous choisissez de redémarrer votre GS avant de configurer ***RamTools,*** il ne se passera rien de spécial, car aucune liste d'outils à charger n'a été enregistrée. Vous remarquerez cependant, en bas et à gauche de votre écran de démarrage, une icône représentant un outil sur une carte mémoire: il s'agit de l'icône de ***RamTools,*** et sa présence indique l'exécution de la séquence de chargement des outils, qui, rappelons-le, ne fait rien puisqu'il n'y a pas de liste d'outils enregistrée.

Pour utiliser le CDev ***RamTools,*** il vous faut activer l'accessoire "Tableau de Bord". Pour ce faire, il faut vous trouver dans un programme sachant traiter correctement les accessoires de bureau du menu , comme le Finder. Déroulez donc ce menu et choisissez l'option "Tableau de Bord". Après quelques instants (la première fois après l'installation ou la désinstallation d'un CDev, c'est un peu plus long, car le Tableau de Bord détecte automatiquement cette modification et reconstruit alors la liste des CDevs présents et actifs qu'il stocke dans le fichier *CDEV.DATA*), vous voyez apparaître la fenêtre du Tableau de Bord, listant dans sa moitié gauche les icônes des différents CDevs dans leur ordre alphabétique. Faites dérouler cette liste avec la barre de défilement jusqu'à voir le CDev ***RamTools*** (son icône est la même que celle affichée lors du démarrage), puis cliquez dessus pour le sélectionner.
Les contrôles de ***RamTools*** sont alors affichés dans la partie droite de la fenêtre du Tableau de Bord. Ils sont au nombre de sept et ont la signification suivante:

❑ Une ***liste de tous les outils*** existant actuellement et correspondant à la version de votre rom couvre la majeure partie de la fenêtre (ne sont proposés que les outils qui ne sont pas déjà en rom). Cette liste est établie à partir de ressources spécifiques de ***RamTools.*** La fin de l'ar- ▶

ticle explique comment modifier ces ressources selon les besoins. C'est à partir de cette liste que vous sélectionnerez tous les outils que vous désirez charger au moment du boot. Pour sélectionner plusieurs éléments de la liste, appuyez sur la touche si ceux-ci sont disjoints, ou sur la touche Majuscules pour en prendre plusieurs d'affilée, tout en cliquant avec la souris.

❑ Le bouton ***Charger*** vous permet de sélectionner automatiquement tous les outils correspondant à la liste déjà enregistrée, par exemple pour la modifier ou simplement pour la visualiser. Ce bouton est désactivé si il n'y a aucune liste d'outils enregistrée.

❑ Le bouton ***Sauver*** réalise l'opération inverse en enregistrant dans les ressources de ***RamTools*** la liste des outils sélectionnés. Les modifications ne sont pas enregistrées systématiquement: si vous fermez ***RamTools***, une alerte vous demandera si vous souhaitez d'abord enregistrer la configuration définie. Dans tous les cas, si une liste d'outils est déjà enregistrée, une alerte vous demandera de confirmer son remplacement si une nouvelle liste est définie, ou sa suppression si aucun outil n'est sélectionné. Ce bouton est désactivé lorsqu'un enregistrement vient d'être effectué, ainsi qu'au démarrage de ***RamTools***, tant qu'aucun outil n'a été sélectionné. L'état des cases à cocher décrites ci-dessous est enregistré d'office par le bouton ***Sauver***, quelque soit votre réponse à l'alerte de confirmation.

❑ Le bouton ***Aucun*** désélectionne tous les éléments de la liste. Il active aussi le bouton ***Sauver***, et vous pouvez alors supprimer la liste déjà enregistrée si elle existe.

❑ Le bouton ***Tous*** sélectionne tous les éléments de la liste en fonction de la case à cocher ***Outils du Bureau.*** Si celle-ci est cochée, seuls les outils utilisés pour piloter le bureau sont sélectionnés; dans le cas contraire, la totalité des outils présents dans la liste est sélectionnée.

❑ La case à cocher ***Outils du Bureau*** est utilisée en conjonction avec le bouton ***Tous*** comme indiqué précédemment. La liste des outils pris en considération par cette option est configurable comme indiqué à la fin de l'article. L'état de cette option est enregistré dans les ressources de ***RamTools*** par le bouton ***Sauver***. Elle est activée d'office dans la version présente sur la disquette ToolBox Mag.

❑ La case à cocher ***Outils chargés au boot*** détermine si la liste des outils enregistrée est effectivement à charger au moment du démarrage. Cette option permet de désactiver temporairement ***RamTools*** sans avoir à annuler la liste des outils enregistrée. L'état de cette option est aussi enregistré dans les ressources de ***RamTools*** lorsque vous utilisez le bouton ***Sauver***. Si vous ne changez que cette option (ou la précédente), il vous faudra charger la configuration enregistrée avant de sauver, sinon la liste actuellement sélectionnée (qui peut être vide) viendra la remplacer (après confirmation) sauf si vous annulez l'alerte, auquel cas seul l'état des deux cases à cocher sera sauvé. Elle est aussi activée d'office dans le programme de la disquette ToolBox Mag.

Un résumé du fonctionnement de ***RamTools*** vous est présenté si vous cliquez sur ***Aide*** pendant que ***RamTools*** est le CDev sélectionné.

Les ressources d'un CDev

Comme indiqué précédemment, pour le programmeur, un CDev se présente sous la forme d'une sous-routine que le Tableau de Bord appelle à un certain nombre de moments déterminés. Plus précisément, un CDev est constitué d'un ensemble de ressources; trois d'entre elles sont obligatoires, ce sont celles référencées par le Tableau de Bord. Les autres, si elles existent, sont spécifiques à chacun des CDevs qui peut donc y mettre ce qu'il veut. Les trois ressources obligatoires sont les suivantes:

❍ La ressource *rIcon* ($8001) d'ID 1 contient l'icône du CDev, telle qu'elle est affichée dans le menu du Tableau de Bord et au démarrage du GS, si le CDev doit être exécuté à ce moment; si c'est le cas, l'icône doit avoir exactement une largeur de 28 pixels. Si l'icône n'est visualisée que par le Tableau de Bord, sa taille n'a pas d'importance, encore que la plupart des CDevs semblent utiliser une taille de 28 x 20 pixels, ce qui assure une certaine homogénéité au menu du tableau de bord; c'est donc la taille que j'ai utilisée pour ***RamTools***.
Cette icône peut être dessinée avec *Genesys,* soit à la main, soit en faisant du copier-coller avec *Platinum Paint*.

❍ La ressource *rCDevFlags* ($8019) d'ID 1 contient un certain nombre d'informations décrivant les actions réalisées par le CDev ainsi que son nom, le numéro de version et le nom de l'auteur, qui sont utilisés dans la fonction d'aide. Nous verrons un peu plus loin le contenu de cette ressource en détail.
Cette ressource peut être créée par *Genesys:* il suffit de remplir les cases.

❍ La ressource *rCDevCode* ($8018) d'ID 1 contient le code du CDev. Il s'agit du programme que vous devez écrire pour im- ▶

planter la fonction désirée du CDev. L'originalité est que ce programme se trouve dans une ressource et ne constitue pas le data fork comme habituellement.
Cependant, sa construction reste classique: il peut être écrit dans votre langage préféré (par exemple ***RamTools*** est écrit en C), puis compilé et linké presque normalement; en effet le programme généré doit être dans le format OMF v2 mais il ne doit pas être au format ExpressLoad (ceci est en général une option du linker, par exemple *ZapLink*, le linker d'ORCA, a une option -x pour ce faire). En assembleur, le programme est tout à fait standard. En C, il ne devra pas y avoir de fonction *main* et il vous faudra supprimer le fichier *.root* avant d'effectuer le link, sauf si vous utilisez ORCA/C version 1.2 qui offre une nouvelle directive empêchant la création de ce fichier (voir plus loin); la fonction appelable doit être la première du source s'il y en a plusieurs. Je pense qu'un CDev peut être écrit avec *ORCA/Pascal* (sans doute pas avec *TML Pascal*), mais j'ignore comment
Le résultat est ensuite intégré sous forme de ressource dans le CDev par la commande *read* de *Rez* ou par *ResDoctor;* il n'y a pas moyen de créer cette ressource avec *Genesys.*
Cette ressource doit impérativement avoir l'attribut *converter*, signifiant au Resource Manager qu'il doit convertir la ressource comme si le programme avait été chargé par le System Loader (d'ailleurs le convertisseur de code appelle *InitialLoad2* avec une option spéciale). Si vous ne le faites pas, le GS va tenter d'exécuter l'entête du segment, et à mon avis, le crash n'est pas loin!

Fonctionnement d'un CDev

Après cette présentation générale, voyons un peu plus en détail le fonctionnement général d'un CDev. Comme il a été dit plus haut, un CDev est une sous-routine du Tableau de Bord, lequel agit donc comme une enveloppe autour du CDev, et réalise la plupart des actions nécessaires que doit effectuer un NDA.
Ainsi, le Tableau de Bord fournit à ses CDevs une fenêtre, qu'ils n'ont donc pas besoin de gérer eux-mêmes, ainsi que la quasi-totalité de l'interaction aussi bien avec l'utilisateur qu'avec le GS. Par exemple, lorsque l'utilisateur clique sur un contrôle d'un CDev, le Tableau de Bord appelle *TaskMasterDA* pour le compte du CDev, lequel fait tout le travail nécessaire, puis indique au Tableau de Bord de quel contrôle il s'agit, etc. Remarquez que, pour l'instant, aucune instruction du CDev n'a encore été exécutée! Cependant, du fait que le contenu de la fenêtre est géré par *TaskMasterDA*, ceci impose qu'elle ne contienne que des contrôles, étendus de surcroît, qui doivent donc être créés avec *NewControl2.* Si un contrôle du CDev a été identifié par le Tableau de Bord, celui-ci appelle éventuellement le CDev en lui passant un message lui demandant de traiter le contrôle.
C'est donc ainsi que la communication se fait entre le Tableau de Bord et un CDev: lorsqu'une des actions prévues par le Tableau de Bord et que le CDev souhaite traiter se produit, le Tableau de Bord appelle le CDev en lui passant un code précisant l'action à effectuer, ainsi que l'objet de l'opération (la fenêtre ou le contrôle par exemple). La version actuelle du Tableau de Bord définit onze de ces messages, dont neuf seulement sont utilisés.

Nous pouvons maintenant expliciter le contenu de la ressource *rCDevFlags* et la structure du programme à intégrer dans la ressource *rCDevCode.*

La ressource *rCDevFlags*

La ressource *rCDevFlags* est constituée de **neuf champs** dont voici la signification:
① **flags**: ce champ occupe un mot et donne son nom à la ressource. Chacun des bits est utilisé pour indiquer au Tableau de Bord si le CDev est intéressé (le bit est à 1) ou pas (le bit est à 0) par chacun des messages susceptibles de lui être envoyés:
✓ les bits 15 à 11 sont réservés pour d'éventuels messages futurs, et doivent être mis à zéro.
✓ le bit 10 *(wantRun)* indique au Tableau de Bord qu'il doit appeler le CDev avec le message *RunCDEV* à chacune de ses exécutions périodiques définies par la période dans l'entête du NDA (actuellement, 1 fois par seconde).
✓ le bit 9 *(wantHit)* demande au Tableau de Bord d'appeler le CDev avec le message *HitCDEV* lorsque l'un des contrôles du CDev a été cliqué.
✓ le bit 8 *(wantRect)* signifie que le Tableau de Bord doit appeler le CDev avec le message *RectCDEV*, avant d'afficher son contenu, pour que celui-ci puisse calculer le rectangle qui sera occupé par ses contrôles. Ceci est utile lorsque, par exemple, le CDev a des contrôles qui varient en fonction de la version de la rom.
✓ le bit 7 *(wantAbout)* indique que le CDev souhaite être appelé avec le message *AboutCDEV* lorsque l'utilisateur clique sur le bouton ***Aide.***
✓ le bit 6 *(wantCreate)* demande au Tableau de



Bord d'appeler le CDev avec le message *CreateCDEV* lorsque l'utilisateur choisit ce CDev dans le menu du Tableau de Bord.
✓ le bit 5 (*wantEvent)* signifie que le Tableau de Bord doit appeler le CDev avec le message *EventsCDEV* juste avant d'appeler *TaskMasterDA*.
✓ le bit 4 *(wantClose)* demande au Tableau de Bord d'appeler le CDev avec le message *CloseCDEV* lorsque celui-ci est fermé, soit en fermant le Tableau de Bord, soit en choisissant un autre CDev.
✓ le bit 3 (*wantInit*) indique au Tableau de Bord qu'il doit appeler le CDev avec le message *InitCDEV* avant d'afficher le contenu du CDev, mais après le message *CreateCDEV* s'il avait été demandé avec *wantCreate*.
✓ le bit 2 *(wantShutDown)*, bien qu'ayant un nom, est réservé et ne provoque pas l'appel du CDev. Il doit être obligatoirement à zéro.
✓ le bit 1 (*wantBoot)* demande que l'init *CDEV.Init* (attention, pour ce message, il ne s'agit pas du Tableau de Bord) appelle le CDev avec le message *BootCDEV* lorsque le GS est démarré.
✓ le bit 0 *(wantMachine)* est réservé et doit donc être à zéro; il pourra ultérieurement provoquer l'appel du CDev avec le message *MachineCDEV*, ce qui permettra, par exemple, au CDev de gérer un périphérique optionnel. Pour l'instant, le Tableau de Bord vérifie automatiquement si le numéro de rom du GS est supporté par le CDev.
② **enabled**: ce champ occupe un octet et indique si le CDev est actif (il vaut alors 1) ou pas (il contient 0).
③ **version:** numéro de version interne du CDev sur un octet, à la discrétion de son auteur.
④ **machine:** numéro minimum de version de la rom que ce CDev requiert (1 octet). Si par exemple, ce champ contient 3, le CDev ne sera pas disponible sur un rom 01.
⑤ **reserved:** 1 octet réservé et devant contenir 0.
⑥ **rectangle:** zone dans laquelle seront dessinés les contrôles du CDev. Son origine doit être en (0,0). Le Tableau de Bord gère la mise en place des contrôles dans le rectangle. Occupe quatre mots.
⑦ **name:** il s'agit d'une *PString* donnant le nom du CDev tel qu'il apparaît sous son icône et dans la fenêtre d'aide. Le nom doit occuper très exactement seize octets en comptant le premier qui est la longueur (si le nom contient moins de 15 caractères, il faudra le compléter avec des espaces).
⑧ **author:** *PString* donnant le nom de l'auteur du CDev, tel qu'il est affiché dans la fenêtre d'aide. Ce champ doit être de trente-trois caractères, avec les mêmes règles que le nom.
⑨ **version:** *PString* d'exactement neuf caractères donnant le numéro de version sous forme ascii; elle est aussi affichée dans la fenêtre d'aide. Ce numéro de version peut être différent de celui défini précédemment sous la forme d'un entier.

La ressource *rCDevCode*

La ressource *rCDevCode* contient donc le code du programme. Comme le Tableau de Bord fait l'essentiel du travail, ce code est en général assez petit.
RamTools est le premier CDev que j'ai écrit; une fois qu'on a compris le mécanisme (qui n'est pas très compliqué), et comme le Tableau de Bord gère la plupart des actions que doit réaliser un NDA, à mon avis, on s'aperçoit qu'*un CDev est ce qu'il y a de plus simple à réaliser en matière de programmation Toolbox*. J'ai écrit ***RamTools*** de façon modulaire, afin qu'il puisse servir de point de départ à vos propres CDevs qui, je l'espère, vont bientôt fleurir dans les colonnes de ToolBox-Mag.
La fonction principale d'un CDev est appelée à la façon d'une routine de la Toolbox et doit donc être déclarée de la manière suivante, selon le langage employé:
❍ En assembleur, la pile se présente ainsi:

Place pour le résultat (4 octets)
Le code du message (2 octets)
data1 (4 octets)
data2 (4 octets)
L'adresse de retour (3 octets)

Le CDev doit dépiler les informations en entrée, placer éventuellement un résultat en fonction du message, et retourner au Tableau de Bord par RTL.
❍ En Pascal, la fonction devra être déclarée ainsi:
Function NomDuCDev (message: Integer; data1, data2: LongInt): LongInt;
❍ En C, la fonction devra être déclarée ainsi:
pascal long NomDuCDev (short message, long data1, long data2) (ORCA/C)
ou
pascal long NomDuCDev (message, data1, data2) (APW C)
short message;
long data1, data2;
La signification de *data1* et de *data2* varie en fonction de chacun des messages. De plus, pour certains de ces messages, le CDev doit retourner une valeur au Tableau de Bord (je n'ai ▶

guère trouvé d'explication claire sur ce point).
D'après l'explication de la ressource *rCDevFlags*, vous pouvez vous attendre à ce que cette fonction soit un unique switch ou case ou JSR (table,x), ce qui est bien entendu le cas!
Voyons donc comment sont utilisés chacun des messages, leurs numéros de code et la signification de *data1* et de *data2:*
✓ ***MachineCDEV (1):*** *Data1* et *Data2* sont inutilisés. Ce message, quand il sera utilisé, est passé au tout début de la sélection du CDev, afin que ce dernier valide son accès en fonction, par exemple, de la configuration matérielle. Si la valeur retournée est 0, alors le CDev est accessible.
✓ ***BootCDEV (2):*** *Data1* et *Data2* sont inutilisés. Le CDev est appelé avec ce message lors du démarrage du GS par l'Init CDEV.INIT et pas du tout par le Tableau de Bord. L'init affiche l'icône du CDev en bas à gauche de l'écran de démarrage pendant son exécution. Aucun outil n'est disponible à ce moment, excepté le Resource Manager. ***RamTools*** utilise ce message pour charger les outils que vous avez sélectionnés et qu'il a enregistrés dans une table stockée dans ses ressources. Notez que du point de vue du GS, *le code du CDev fait effectivement partie d'une init*, c'est pourquoi l'astuce de chargement des outils marche aussi.
✓ ***ShutDownCDEV (3):*** *Data1* et *Data2* ne sont pas définis, de même que le moment où ce message est passé au CDev.
✓ ***InitCDEV (4):*** *Data1* contient le pointeur sur la fenêtre du Tableau de Bord et *Data2* est inutilisé. Ce message est passé au CDev après le *CreateCDEV*, mais avant que les contrôles soient dessinés. Le CDev peut positionner ses contrôles à leurs valeurs initiales; c'est ce que fait ***RamTools*** qui construit la liste des outils sélectionnables, positionne les deux cases à cocher et invalide le bouton *Sauver* et éventuellement *Charger*.
✓ ***CloseCDEV (5):*** *Data1* et *Data2* sont inutilisés. Le CDev est appelé avec ce message lorsqu'il est fermé, soit parce que le Tableau de Bord est lui-même fermé, ou parce que l'utilisateur a choisi un autre CDev. Le CDev peut alors enregistrer les valeurs des contrôles, disposer de la mémoire qu'il a allouée spécifiquement (la mémoire allouée pour les contrôles est libérée par le Tableau de Bord), etc. C'est ce que fait ***RamTools*** lorsqu'il reçoit ce message, après confirmation de l'utilisateur en ce qui concerne la sauvegarde. Il en profite aussi pour arrêter les outils qu'il a démarrés à l'étape *CreateCDEV*.
✓ ***EventsCDEV (6):*** *Data1* contient un pointeur sur un *Event Record* non étendu (il ne contient pas les infos gérées par TaskMaster) et *Data2* est inutilisé. Ce message est passé au CDev juste avant que le Tableau de Bord appelle *TaskMasterDA*, ce qui lui permet, par exemple, de modifier l'*Event Record* à sa guise, ou de faire un traitement supplémentaire. ***RamTools*** n'utilise pas ce message.
✓ ***CreateCDEV (7):*** *Data1* contient le pointeur sur la fenêtre du Tableau de Bord et *Data2* est inutilisé. Ce message est passé au CDev lorsque l'utilisateur le choisit dans la liste d'icônes; il doit alors créer ses contrôles. Le plus simple est de les définir sous forme de ressources et de faire un appel à *NewControl2*. C'est la méthode employée par ***RamTools***; cependant, comme il utilise un *List Control*, et que le List Manager n'est pas forcément démarré (TextEdit est aussi utilisé, comme nous allons le voir juste après), ***RamTools*** charge et démarre tous les outils dont il a besoin à l'aide d'une version très légèrement modifiée de la routine (dans sa version C) *StartNDATools* publiée dans ToolBox Mag 5 [j'ai rajouté un paramètre indiquant si l'on veut démarrer le Resource Manager ou pas, car dans un CDev, il ne faut surtout pas le démarrer – c'est fait pour vous par le Tableau de Bord –, sinon lors de l'appel à *NewControl2*, vous allez tout de suite voir la défiler sur votre écran texte, avec un message d'erreur $1E06 (ressource non trouvée); c'est ce qui m'est arrivé avant que je fasse la modification]. Vous trouverez, bien entendu, le source modifié sur la disquette ToolBox-Mag.
Une fois que les outils ont été démarrés, il ne reste plus qu'à appeler *NewControl2* en lui passant une ressource *rControlList* ($8003) contenant la liste de tous les contrôles stockés dans des ressources *rControlTemplate* ($8004). ***RamTools*** récupère enfin le numéro de version de la rom (à l'aide d'un petit morceau d'assembleur) qu'il utilise dans la partie *InitCDEV* pour déterminer si un outil est déjà en rom ou doit faire partie de la liste.
✓ ***AboutCDEV (8):*** *Data1* contient le pointeur sur la fenêtre d'aide du Tableau de Bord et *Data2* est inutilisé. Ce message est passé au CDev lorsque l'utilisateur clique sur le bouton *Aide*. Le Tableau de Bord a déjà créé la fenêtre, il y a placé l'icône du CDev, son nom, son numéro de version, le nom de son auteur, et le bouton *OK* qu'il gère automatiquement. Il ne reste plus au CDev qu'à placer le texte d'aide; le plus simple est d'utiliser un contrôle de type *statTextControl* qui lui-même fait référence à une ressource de type *rTextForLETextBox2* ($800B) contenant le texte d'aide, et de faire ▶

appel encore une fois à *NewControl2* pour l'afficher (c'est la technique utilisée dans les CDevs standard). ***RamTools*** procède un peu différemment, car le texte d'aide est trop important pour l'espace disponible; un contrôle étant un contrôle, j'ai utilisé un TextEdit (c'est pour cela qu'il est démarré dans *CreateCDEV*, ainsi que les outils qu'il requiert: Font Manager et QuickDraw Auxiliary), faisant référence à une ressource de type *rText* ($8016); c'est le type de ressource utilisé d'office par *Genesys*, et je ne suis pas arrivé à utiliser une ressource *rTextForLETextBox2* avec *Genesys* (*Genesys* n'est pas content, et le Tableau de Bord n'affiche rien); l'inconvénient est qu'il n'est alors pas possible de formater le texte à sa convenance.

✓ ***RectCDEV (9):*** *Data1* pointe sur le rectangle défini dans la ressource *rCDEVFlags* et *Data2* est inutilisé. Ce message peut être utilisé par un CDev pour ajuster son rectangle d'affichage (la taille maximale étant de 200 x 175 pixels), par exemple, si la liste des contrôles varie en fonction de la configuration. Ce message n'est pas utilisé par ***RamTools*** qui exploite la totalité de l'espace disponible.

✓ ***HitCDEV (10):*** *Data1* est un handle sur le contrôle cliqué et *Data2* son ID. Ce message est bien entendu passé au CDev lorsque l'un de ses contrôles a été cliqué. Je ne vais pas vous détailler les actions de ***RamTools*** pour chacun de ses contrôles, car tout ceci est assez trivial; je vous renvoie au source pour de plus amples informations, les particularités sont largement commentées.

✓ ***RunCDEV (11):*** *Data1* et *Data2* sont inutilisés. Ce message correspond à l'exécution périodique d'un NDA, et dans le cas du Tableau de Bord, elle a lieu une fois par seconde. Il peut être utilisé, par exemple, par un CDev de type horloge. ***RamTools*** ne l'utilise pas.

Notes de Programmation

RamTools a été développé avec Orca/C, dans sa version 1.2. Si vous utilisez encore Orca/C 1.1, je vous conseille de faire la mise à jour, car elle corrige surtout tous les bugs de l'optimisation, ce qui la rend utilisable dans un programme de bonne taille; elle apporte aussi quelques améliorations comme le pragma *noroot* qui permet de ne pas générer le fichier *.root* inutilisé dans un CDev. Si donc vous ne l'avez pas encore, vous devrez détruire ce fichier avant d'effectuer le link (voir le fichier *Make* décrit plus loin); vous pouvez laisser le pragma qui sera ignoré par le compilateur 1.1. Je vous conseille aussi de supprimer les optimisations *(#pragma optimize 0)* si vous ne voulez pas vous exposer à des surprises désagréables; de toute manière, cela ne change pas grand chose aux performances ou à la place occupée dans le cas présent.

Si vous utilisez APW/C et que vous souhaitez recompiler le programme, il vous faudra le modifier sensiblement, en particulier au niveau des déclarations des fonctions. Au moment du link, je pense que vous ne devez pas faire référence à START.ROOT parce qu'il n'y a pas de fonction *main*, et surtout pas besoin d'un appel à *Quit*. Il vous faudra aussi faire attention à la taille du programme final, car *une ressource de code ne doit comprendre qu'un seul segment, et est donc limitée à 64 KO,* mais j'espère que dans le cas présent APW C ne linkera pas toute sa librairie, sachant que je n'ai n'en quasiment utilisé aucune fonction.

Comme je l'ai déjà indiqué, la plupart des ressources ont été éditées avec *Genesys v1.2;* toutes celles qui sont utilisées directement par le programme ont été nommées, pour qu'il soit plus simple de s'y retrouver. Vous trouverez sur la disquette ToolBox-Mag le source *Rez*, ainsi que les définitions des noms des ressources, tel qu'il ont été générés par Genesys, et copieusement modifiés ensuite. Bien que Genesys permette de créer des contrôles indépendamment d'une fenêtre, ce n'est pas une méthode que je vous recommande (je m'en suis rendu compte après coup), car il ne donne alors pas la possibilité de définir la position et la taille des contrôles, ainsi que d'autres caractéristiques, comme la taille des membres de la liste qui vaut 9, au lieu de 5 habituellement.

J'ai conçu ***RamTools*** de la manière la plus évolutive possible, en stockant l'ensemble de ses paramètres dans des ressources. Il pourra ainsi s'adapter facilement, en principe sans même modifier le source, aux évolutions futures du GS.

Pour ce faire, j'ai utilisé trois types de ressources spécifiques, et faisant preuve de non-originalité, j'ai pris les numéros 1, 2 et 3. Je les ai cependant baptisés en fonction de leur utilisation:

① Le type 1, *rInteger*, comme son nom l'indique contient un entier. Il y a quatre ressources de ce type:

✓ La ressource d'ID 1, *LoadToolsAtBoot*, contient le flag indiquant si la table d'outils doit être chargée (la valeur 1 est stockée dans la ressource) ou pas (la ressource contient un 0) lors du démarrage du GS.

▶

✓ Les ressources d'ID 2, *MinToolNum*, et 3, *MaxToolNum*, contiennent les numéros mini et maxi des outils décrits dans les ressources de type 2.
✓ La ressource d'ID 4, *DesktopToolsOnly*, contient la valeur du flag associé à la case à cocher *Outils du bureau*.
② Le type 2, *rToolDesc*, décrit chacun des outils sélectionnables dans la liste proposée par ***RamTools*** et pouvant être chargés lors du boot. Il y a une ressource de ce type par outil, chaque ID devant être égal au numéro d'outil représenté par la ressource. Les ressources de type *rInteger MinToolNum* et *MaxToolNum* délimitent les ID utilisés. Si vous ajoutez de nouvelles descriptions d'outils, il vous faudra aussi éventuellement modifier la valeur maxi. L'édition de ces ressources ne peut se faire qu'avec *Rez* ou *ResDoctor*. Chaque ressource de ce type occupe vingt octets répartis en *trois champs:*
❶ Une *PString* de seize caractères (plus l'octet de longueur) donnant le nom de l'outil. Si moins de seize caractères sont utilisés, il vous faudra compléter avec des espaces.
❷ Le dix-huitième octet contient des flags, dont les bits ont la signification suivante:
✓ les bits 7 à 5 sont inutilisés et doivent être à 0.
✓ le bit 4 est à 1 si cet outil est utilisé pour piloter le bureau et à 0 dans le cas contraire. C'est ce bit qui est utilisé par la case à cocher *Outils du bureau* et le bouton *Tous*.
✓ les bits 0 à 3 donnent le numéro de version maxi de la rom pour prendre en considération cet outil, ce qui permet d'aller jusqu'à une version de rom 15. S'ils contiennent 0, l'outil est ignoré (c'est la valeur que j'ai mise pour l'outil 33, Video Overlay, dont peu d'entre nous ont besoin); s'il y a 1, l'outil n'est à proposer que sur un rom 01 (il est en rom sur un rom 3); si la valeur est 3, alors cet outil doit être pris en considération pour toutes les versions de la rom. Si jamais, il y a un jour un GS rom 4, il faudra modifier cet octet en conséquence.
La détermination des outils déjà en rom sur un rom 3 a été faite de mémoire, car je n'ai pas retrouvé l'information, et je me suis sans doute trompé: si la liste qui vous est proposée est incorrecte, vous savez où il faut corriger.
❸ Les dix-neuvième et vingtième octets contiennent le numéro de version de l'outil, tel qu'il doit être passé à *LoadTools*. J'y ai mis les versions correspondant au système 5.0.4, afin d'assurer le chargement des bonnes versions. Si l'un d'entre eux manque à l'appel lors d'une vérification post-démarrage, c'est que vous avez mal installé votre système 5.0.4 (par exemple, s'il ne vous manque que QDAux 3.3, c'est que vous êtes en 5.0.3).
③ Le type 3, *rToolTable*, ne contient qu'une seule ressource, d'ID 1, qui est la table d'outils à charger lors du démarrage. Son format est très exactement celui attendu par *LoadTools*, à savoir le nombre d'outils dans la table, et pour chacun d'entre eux, son numéro et le numéro de version minimal pour le charger. Elle n'est pas présente dans le fichier de la disquette ToolBox-Mag. Elle est créée lorsque vous cliquez sur le bouton *Sauver*. Si à ce moment, aucun outil n'est sélectionné, la ressource, si elle existe, est supprimée.

Voilà, je pense avoir tout dit. Pour le reste, je vous renvoie au source, qui je pense est assez clair, même si vous ne connaissez pas le C. La programmation d'un CDev est ultra simple; j'ai même passé plus de temps à écrire cet article que le programme! Sur la disquette ToolBox-Mag, vous trouverez les fichiers suivants dans le dossier /RAMTOOLS.CDEV:
• *RamTools.cc:* le source Orca/C du CDev.
• *RamTools.h:* définition des constantes, types et variables globales utilisés dans *RamTools.cc*.
• *RamTools.rez:* le source *Rez* des ressources de ***RamTools***, tel qu'il a été généré par *Genesys*, et corrigé ensuite.
• *RamTools.rez.h:* contient les définitions des noms des ressources; il a aussi été généré par *Genesys* (option *Generate Equates*). J'ai rajouté son *#include* dans *RamTools.rez*.
• *StartTools.cc* et *StartTools.h:* la nouvelle version de *StartNDATools* et *StopNDATools* dont j'ai parlé plus haut.
• *RamTools.r:* fichier de ressources édité avec *Genesys*. Il contient la totalité des ressources du CDev à l'exception du code.
• *RamTools:* c'est le CDev que vous devrez recopier dans le dossier /SYSTEM/CDEVS de votre disque de démarrage.
• *Make:* fichier exec reconstruisant l'ensemble du programme avec Orca/C et Rez. Il est prévu pour être utilisé avec Orca/C 1.2, une petite adaptation (indiquée dans le fichier) est à effectuer si vous utilisez la v1.1 ou APW/C.

La partie de l'article concernant la programmation d'un CDev (et l'écriture du programme!) est fondée sur la note de description de type fichier *FTN.C7.xxxx*, qui est, à ma connaissance, la seule référence officielle sur le sujet. L'essentiel des informations a été repris dans cet article. J'ai aussi bien sûr utilisé la Bible, à savoir les trois volumes de l'indispensable *Toolbox Reference Manual*, ainsi que le *GS/OS Reference Manual*. ■

La Laser sans corset

J.Y. Bourdin

Cet article est la suite de "La Laser sans le Mac" de ToolBox-Mag 5. Au départ, je n'avais pour but que d'enfoncer un peu le clou à propos de la différence entre PostScript et TrueType (avec la sortie du système 7 du Mac, la publicité d'Apple nous "brouille l'écoute" à ce sujet), en présentant quelques démonstrations des possibilités de la Laser PostScript. Vous trouverez ces démonstrations dans les pages qui suivent, et les programmes correspondant sur la disquette: le premier qui nous envoie les mêmes pages sous TrueType gagne un abonnement gratuit à ToolBox-Mag pour la personne de son choix...

Mais ce dont je me suis très vite aperçu, en mettant au point ces programmes à partir d'exemples de manuels, c'est que de toute façon, dès qu'on veut faire des choses un peu sérieuses avec la Laser, il faut lui parler directement dans son langage, le PostScript. Avec le corset de QuickDraw, sur GS comme sur Mac, au moins 80% des possibilités de la Laser sont inexploitées.
Or, pour parler en PostScript à la Laser, n'importe quel ordinateur marche: GS, Mac, IBM ou n'importe quoi.

Communiquer avec la Laser

Je rappelle que vous avez deux moyens pour communiquer avec la Laser depuis le GS. Soit vous passez par Appletalk, et il faut alors lui envoyer des fichiers textes pseudo-IWem, stockés dans /SYSTEM/DRIVERS, et envoyer ces pseudo-IWem par le CDEV LaserWriter. Il faut simplement avoir un accessoire permettant de changer à la volée les noms de fichiers dans ce catalogue. Voyez ToolBox-Mag 5.
Ce système n'étant pas très confortable, vous pouvez choisir l'autre: connecter le GS par le port série. Dans ce cas, le GS dispose d'**un logiciel extraordinaire pour communiquer avec la Laser: AppleWorks-GS**, tout simplement. Merci à Jean-Luc Schmitt, qui nous avait déjà expliqué dans Pom's combien AppleWorks-GS était confortable pour faire de la communication entre deux ordinateurs. Or, précisément, je le rappelle, la Laser n'est pas une imprimante, mais un ordinateur.
Finalement, mon GS est **connecté deux fois** à la Laser, par le port série et par AppleTalk. Sous AppleWorks-GS, le dialogue avec la Laser est ultra-facile, et je choisis tantôt le module de communication et le port série, tantôt AppleTalk et le driver de GS/OS, le tout sans jamais rebooter, sans avoir rien à sauver sur disque.
Sauf dans un cas: quand le fichier tel que l'imprime le driver Apple ne me plaît pas, je l'imprime sur disque. Ensuite, j'édite ce fichier dans le traitement de textes d'AppleWorks-GS. S'il est un peu long, je l'envoie comme pseudo-IWem. S'il n'est pas trop long, copier-coller dans le module de communication, et en direct sur la Laser.
Si vous transportez les fichiers PostScript de la disquette ToolBox-Mag sur une disquette MS-Dos ou HFS/Mac, ils marcheront aussi sur IBM ou sur Mac. Ce sera juste nettement plus la galère pour communiquer avec la Laser.
AppleWorks-GS, l'intégration aussi souple du module de communication avec les autres modules, ça n'existe pas sur Mac ou sur IBM: Works y fait pâle figure, et les MultiFinder de Windows 3 et du Mac ont l'air de succédanés pour pauvres, comparés à l'intégration d'AppleWorks-GS.
Je préfère le confort et la liberté du GS, mais chacun ses goûts (et chacun ses contraintes professionnelles)...

Trucs et découvertes

En dialoguant avec la Laser, on découvre d'étranges choses. Par exemple, c'est le premier ordinateur que je connais où on ait **laissé les Rems dans un programme en Rom**. Oui, des commentaires dans la Rom. Prenez par exemple le programme *StartPage* du dictionnaire *UserDict:* c'est un programme PostScript ordinaire, avec toutes ses remarques, y compris un bel encadré pour le titre (en PostScript, tout ce qui suit le signe % est un commentaire, que l'interpréteur néglige). Si vous voulez apprendre le PostScript, la Rom de la Laser va vous aider pas mal...
De même, les fontes *EmulatorFont* et *EmulatorFont-Bold* de la Rom de la Personal NT sont parfaitement accessibles et utilisables en PostScript. Vous trouverez sur la disquette un programme qui les imprime. Pour qu'elles soient utilisables avec les drivers Apple, il aurait fallu quelques lignes de PostScript dans les drivers pour les réencoder, et fabriquer les fontes-écran QuickDraw correspondantes. Même pour le Mac, Apple n'a pas daigné faire ce travail. Quand je parlais d'un service en-dessous de tout dans le numéro 5, ça ne se limitait pas au GS...

Page publicitaire ou page de test ?

La première chose à faire en dialoguant avec votre Laser, c'est de lui dire de cesser d'imprimer une page publicitaire à chaque fois qu'on l'allume ou qu'on lui envoie un Reset. Pourquoi? Eh bien, parce que la seule manière de récupérer la mémoire de la Laser après que les drivers Apple y aient déposé di- ▶

verses choses, c'est de faire un Reset. Vous allez donc devoir faire fréquemment des Reset: ne gâchez pas temps, toner et papier en imprimant à chaque fois une feuille pour rien. Voyez ci-dessous le paragraphe sur la gestion de la mémoire.
Pour désactiver cette page publicitaire automatique, un programme de deux lignes suffit. Le voici:
serverdict begin 0 exitserver
statusdict begin false setdostartpage end
Pour remettre la page publicitaire automatique, il suffit d'envoyer le même programme, en remplaçant *false* par *true* dans la deuxième ligne.

Vous pouvez désormais envoyer des Reset pour récupérer la mémoire de votre Laser, qui ne prend plus pour cela qu'une quinzaine de secondes au lieu de presque une minute. Les deux lignes suivantes suffisent pour faire un Reset:
serverdict begin 0 exitserver
systemdict /quit get exec end

Bon, mais, me direz-vous, il y avait une information intéressante dans cette page publicitaire: le compteur qui indique le nombre de pages imprimées depuis le début. Ce que je vous propose, c'est de remplacer cette page publicitaire par une page technique **infiniment plus intéressante**, la **page de test**. Vous saurez ainsi non seulement le chiffre du compteur, mais aussi des choses beaucoup plus cachées, par exemple le numéro de série de votre Laser, ses réglages, l'occupation de sa mémoire, etc. Pour imprimer cette page, il suffit d'envoyer le mini-programme suivant (quatre mots!):
serverdict begin printtestpage end

Le mot de passe

Il y a un mot de passe dans votre Laser, qui est vierge au départ (c'est 0, zéro). Si vous vous dîtes que vous avez tout le temps de vous occuper de ça plus tard, vous avez tort. **Si ce n'est pas vous qui mettez le mot de passe, quelqu'un d'autre va vous le mettre!** Et après cela, vous ne pourrez plus le changer, et vous ne pourrez plus rien changer aux réglages de votre Laser! Profs qui travaillez sur Laser, sachez qu'**il y a des élèves qui lisent ToolBox-Mag**, et dépêchez-vous.
Car deux lignes de programme suffisent pour remplacer le mot de passe par défaut (zéro) par "1":
serverdict begin 0 exitserver
statusdict begin 0 1 setpassword end
A la place de "1", vous mettez bien entendu ce que vous voulez, la syntaxe de *setpassword* étant *AncienPasse NouveauPasse setpassword.*
Attention, il faudra ensuite que tous les programmes de réglages ou de Reset qui commençaient par *serverdict begin 0 exitserver* mettent votre nouveau mot de passe à la place du zéro.

Un programme PostScript

En guise d'initiation à PostScript, j'ai choisi de commenter un petit morceau d'un des programmes de démonstration dont vous voyez les résultats dans les pages suivantes, celui qui écrit "ToolBox-Mag" en rayons dans tous les sens. A la différence des programmes précédents, qui ne marchent que sur LaserWriter Apple, celui-ci doit marcher sur toute imprimante Laser PostScript.
Une remarque générale pour pouvoir lire ce programme: PostScript est un de ces langages qui travaillent avec la pile, genre Forth. Cela a une conséquence immédiate: les expressions PostScript se lisent **de droite à gauche.** Par exemple l'expression
NiveauGris 0.05 sub
se lit: soustraire (*sub*) 0,05 à *NiveauGris*, le résultat étant à son tour placé sur la pile.

L'avantage de PostScript, c'est l'extraordinaire liberté qu'il donne dans la programmation comme dans le graphisme.
Les deux difficultés de PostScript sont dans les deux questions qu'il faut en permanence se poser quand on programme en PostScript:
- **Où en est la pile** (du point de vue de la programmation)?
- **Où suis-je** (du point de vue du graphisme)?

Du point de vue de la structure, un programme PostScript peut se présenter... absolument n'importe comment. La seule exigence, c'est que les objets aient été définis avant leur utilisation.
Pour nous y repérer un peu mieux, et en hommage à J.Destelle, j'ai présenté ce programme 

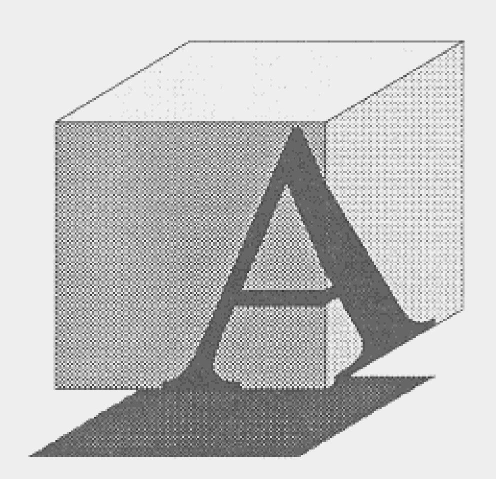
```
%----------  Préambule  -----
/Memoire save def
initgraphics

%-----  Déclarations et procédures  ------------
/Cm          { 28.35 mul } def
/AngleRot  155 def
/Decalage  -22.5 def
/NiveauGris  0.9 def
/Origine     { 2.5 Cm 6.8 Cm moveto } def
/BoucleRot {gsave
               Origine
               AngleRot rotate
               NiveauGris setgray
               (ToolBox-Mag) show
               /AngleRot AngleRot Decalage sub def
               /NiveauGris NiveauGris 0.05 sub def
             grestore} def

%-------  Programme Principal  ------------
/Helvetica findfont [ 24 0 0 24 0 0 ] makefont setfont
gsave
  0 0 translate 5 Cm 7.5 Cm translate
  0.75 0.75 scale
  20 { BoucleRot } repeat
grestore
initgraphics

%------------  Conclusion  -----------
showpage
Memoire restore
```









Fonte Emulator

"à la Pascal": d'abord les constantes et variables, puis les "procédures" (en l'occurrence la sous-routine d'impression), puis le "programme principal", qui ne fait guère qu'appeler la "procédure". Cette structure n'est pas du tout obligatoire: le programme tout entier peut être présenté comme une procédure, un sous-programme d'un autre programme.

Gestion de la mémoire

La gestion de la mémoire en PostScript (Niveau 1 du moins) est plus que fruste. Seule la paire d'instructions *save / restore* peut être utilisée: au début de son programme, on sauve (*save*) sur la pile l'état de la mémoire tel qu'on le trouve au début, on le restaure (*restore*) à la fin.

[C'est d'ailleurs pourquoi je conseille de désactiver l'impression de la page de publicité lors d'un Reset: en-dehors de l'usage de save / restore, il n'y a pas d'autre moyen de récupérer la Ram qu'un Reset sauvage. C'est en particulier la seule manière de nettoyer les dépôts encombrants de dictionnaires et bouts de programmes divers que les drivers et les applications laissent sans vergogne dans la Ram de la Laser. C'est aussi le seul moyen de se débarrasser d'une fonte déposée à demeure. Puisqu'on est fréquemment amené à faire un Reset, inutile de gâcher du temps, du toner et du papier.]

Voilà pourquoi la première instruction de notre programme sauvegarde l'état de la Ram dans un objet appelé *"Memoire"*, tandis que la dernière instruction restaure cet état.

Tant que nous y sommes, disons un mot de l'avant-dernière instruction de notre programme, *showpage*. C'est elle qui fait imprimer effectivement sur le papier. Tant que cette instruction n'apparaît pas, on trace en mémoire, sur une page virtuelle. On se déplace librement sur cette page virtuelle, on peut superposer des dizaines de tracés, de lignes, etc.

Voilà pourquoi il est si facile de faire du couper/coller en PostScript: on peut toujours tracer quelque chose en plus, tant que PostScript n'a pas lu *showpage* il rajoute des tracés. PostScript étant un langage *interprété* (contrairement au Pascal destellien), il exécute au fur et à mesure les instructions qu'il lit, et n'imprime rien effectivement tant qu'il n'est pas arrivé à *showpage*.

Les objets PostScript

On notera dans la première ligne du programme l'usage de l'instruction *def*. C'est une des très grandes forces de PostScript: n'importe quoi peut devenir un objet PostScript, il suffit de l'enclore entre deux accolades (sauf si la définition ne contient qu'un élément) et de l'encadrer au début par le nom qu'on donne à l'objet (tout ce qui commence par / est un nom), et à la fin par l'instruction *def*, sur le modèle: */Maroutine { mes instructions } def*

Tout, absolument tout, peut devenir un sous-programme ou une procédure d'un autre programme. Rien ne vous empêche par exemple d'inclure tout le programme ci-dessous dans une définition d'objet PostScript, et de l'utiliser comme instruction dans un autre programme. Vive le Couper-Coller!

Regardez par exemple l'usage de l'objet *Cm:* les unités de départ en points PostScript n'étant pas très parlantes, on définit une opération «multiplier par 28,35», ce qui s'écrit *28.35 mul*, dont le résultat est de convertir les points PostScript en centimètres. Mettez le tout entre accolades, et l'objet *Cm* (qui est une opération) s'utilisera désormais comme une nouvelle unité dans toutes les instructions qui demandent des mesures de distance, la conversion étant faite automatiquement. Cette unité est par la suite utilisée sans problème pour définir l'objet *Origine*, qui décrit un déplacement relatif de 2,5 centimètres sur l'axe des x et 6,8 centimètres sur l'axe des y (*2.5 Cm 6.8 Cm moveto*).

Gestion de l'état graphique

La seconde instruction du programme, *initgraphics*, est une sorte de Reset de l'état graphique, pour savoir à peu près de quoi on part.

Le programme lui-même commence par des définitions de constantes et de variables, incluant des opérations comme le *Cm* dont nous avons déjà parlé. *Decalage* et *Origine* sont des constantes, qui seront utilisées ensuite sans modification. *NiveauGris* et *AngleRot*, en revanche, sont des variables. Il n'y a en fait aucune différence entre constante et variable, du point de vue de PostScript. *NiveauGris* et *AngleRot* étant redéfinis à chaque passage dans la boucle, ce sont des variables.

C'est la boucle *BoucleRot* qui va faire tout le travail de notre programme. Elle commence par une instruction *gsave* et se termine par son correspondant *grestore*. Ces instructions sont à l'état graphique de PostScript ce que *save* et *restore* sont à la gestion de la mémoire. Beaucoup d'instructions graphiques doivent être encadrées par ce couple, sans quoi, particulièrement avec les instructions qui modifient le système de coordonnées, comme *translate* et *rotate*, on finit par ne plus savoir du tout où on est. C'est pourquoi vous retrouvez ce couple *gsave/grestore* encadrant la partie principale du programme.

Descartes et la Laser

Un des principaux auteurs de PostScript, à l'évidence, est René Descartes. Non seulement Descartes savait déjà la différence entre l'homme et l'ordinateur ("Je pense donc je suis"), mais il savait aussi ce qu'on peut faire faire à un ordinateur (calculer): c'est lui l'inventeur de ces fameux "repères cartésiens" (axe des x, axe des y, vous vous rappelez ?) qui lui ont permis de montrer que la géométrie, en fait, c'était de l'algèbre, et les courbes, rien d'autre que des fonctions. Or les fonctions, cela passe comme lettre à la poste dans l'ordinateur: tracer, c'est calculer.

Ne cherchez pas pourquoi le point 0,0 est situé en **bas à gauche** de la feuille, en PostScript, et pas en haut à gauche comme on en a l'habitude: c'est ▶

ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle; avec TrueType, essayez donc! ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle; avec TrueType, essayez donc! ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle; avec TrueType, essayez donc! ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle; avec TrueType, essayez donc! ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle; avec TrueType, essayez donc! ToolBox-Mag - exemple d'utilisation d'une fonte PostScript avec changements continus de taille et d'angle;

Les merveilles de makefont :
[12 0 0 12 0 0] makefont
[12 0 .5 12 0 0] makefont
[12 0 -.5 12 0 0] makefont
[9 0 0 12 0 0] makefont
[7 0 0 12 0 0] makefont
[15 0 0 12 0 0] makefont
[19 0 0 12 0 0] makefont
[12 -4 -4 12 0 0] makefont

Ces exemples n'illustrent que quelques talents de PostScript ayant trait aux fontes, aux rotations, et aux niveaux de gris. Tout cela devrait donc pouvoir se faire aussi avec TrueType. On en est loin. Il ne suffit pas de le vouloir pour remplacer PostScript !

▶

simplement que cela permet de travailler avec des valeurs toujours positives pour x et y.
La véritable magie des repères cartésiens est cependant dans la suite: au lieu de vous embêter à tracer des droites formant un angle de 33° avec l'axe des y, tracez une droite bien parallèle à cet axe: il suffit d'avoir fait tourner les axes de coordonnées de 33°! C'est cela que fait ce programme: l'instruction d'impression *(ToolBox-Mag) show* est strictement équivalente à *PRINT "ToolBox-Mag"* en Applesoft. Aucun de ces "ToolBox-Mag" en rayons de vélos n'est imprimé en biais ou de façon tordue. **Tout est imprimé tout droit, horizontalement. Simplement, auparavant, on a fait tourner d'un cran la feuille de papier!**

Le restant ce ce que fait cette sous-routine est enfantin: *Origine* nous place au point fixé par cette constante (dans les coordonnées en cours, attention); *AngleRot rotate* fait tourner les axes de cordonnées (la feuille) de *AngleRot. NiveauGris setgray* fixe la couleur (le niveau de gris) avec lequel on va imprimer; *(ToolBox-Mag) show* imprime "ToolBox-Mag". On redéfinit ensuite la variable *AngleRot* (*/AngleRot ... def*) en soustrayant la valeur de *Decalage* de la valeur actuelle de *AngleRot* (*AngleRot Decalage sub*): si bien qu'au prochain tour l'angle de rotation aura été décalé d'un cran. De la même façon, on assombrit de 5% le niveau de gris pour le prochain tour en soustrayant 0,05 de la valeur actuelle de *NiveauGris* (*/NiveauGris NiveauGris 0.05 sub def*). Les niveaux de gris sont échelonnés entre 1 (blanc) et 0 (noir).

Traitons tout de suite de la partie "Programme Principal", qui se contente de répéter 20 fois la boucle *BoucleRot (20 { BoucleRot } repeat)*, mais après avoir utilisé deux autres instructions "cartésiennes" de PostScript, *translate* et *scale*.
Si l'origine de PostScript en bas de la page vous gêne, ou surtout si vous voulez travailler avec des chiffres simples, c'est facile: **déplacez l'origine!** C'est ce que fait l'instruction *translate* (*5 Cm 7.5 Cm translate*). Si j'ai mis un *0 0 translate* avant, c'est que, pour opérer une translation, il est bon de savoir d'où on part. On risque très vite de ne plus savoir où on est.
Même facilité pour l'échelle: vous voulez travailler avec des nombres simples, et pas d'atroces fractions, c'est tout simple: **changez l'échelle!** C'est ce que fait l'instruction *0.75 0.75 scale*. Pourquoi deux valeurs pour cette instruction *scale* ? C'est que rien ne vous oblige à choisir le même facteur d'échelle pour l'axe des x et pour l'axe des y. Vous pouvez étirer la feuille en hauteur sans l'élargir, et vice-versa.

Les fontes normales, ça n'existe pas.

Ce programme comprend une ligne pour choisir la fonte utilisée. Elle se lit de la façon suivante:
/ Helvetica findfont = trouve la fonte Helvetica;
[24 0 0 24 0 0] makefont = fais-en une fonte de 24 points;
setfont = fais-en la fonte courante.
L'instruction intéressante est *[24 0 0 24 0 0] makefont*: elle montre qu'il faut donner au moins **deux** valeurs pour définir la taille d'une fonte. Descartes est encore passé par là. Les six valeurs qui précèdent, entre crochets, l'instruction *makefont*, constituent une matrice. La première et la quatrième valeur sont l'échelle sur l'axe des x et sur l'axe des y. La seconde et la troisième contrôlent respectivement l'angle de l'axe des x et celui des y. La cinquième et la sixième contrôlent l'origine des x et celle des y. Cela montre que nous avons le choix complet, en hauteur, en largeur et en orientation pour les fontes.
Le driver de LaserWriter Apple fait des modifications d'échelle, mais sans prévenir, il appelle cela les modes "condensé", "à la Macintosh", et autres dénominations fantaisistes. Si je qualifie ces dénominations de fantaisistes, c'est que le "mode Macintosh" du GS n'est pas identique à l'impression sur un Macintosh, et que "mode Macintosh" ne veut strictement rien dire en PostScript (la seule chose qui pourrait être spécifique au Macintosh serait le vecteur d'encodage des fontes, mais il ne s'agit pas de cela dans le dialogue du driver LaserWriter).
On ferait mieux de nous proposer clairement le choix des facteurs d'échelle: puisque de toute façon il n'y a pas de "taille normale" des fontes, et qu'il faut faire des choix, autant donner ces choix à l'utilisateur.
Qu'est-ce qui l'empêche? Tout simplement *QuickDraw:* aussi bien sur Mac que sur GS, QuickDraw est bien incapable de représenter sur l'écran toutes ces transformations. Par exemple, imprimez quelque chose sur Laser avec un Helvetica dont la matrice *makefont* serait *[80 140 0 110 0 0]*. Joli effet, n'est-ce pas? Eh bien, maintenant, essayez donc d'en donner la représentation écran avec QuickDraw, sur GS ou sur Mac...
Si vraiment ces limites vous gênent, lisez l'article de Jean-Michel Vallat dans ce numéro: si on veut faire du *vrai* "Wysiwyg", la seule solution est **Display PostScript**. Tout le reste n'est que bidouilles diverses et mensonges par omission auprès des utilisateurs, à qui on cache soigneusement qu'il existe bien d'autres choix que ceux qu'on leur propose (à moins qu'on ne leur présente comme un miracle inouï la possibilité d'avoir des fontes "étroitisées" ou "élargies", ce qui est le b–a–ba du PostScript).

Ça, c'est PostScript.

On le voit, tout notre programme consiste essentiellement en transformations des axes de coordonnées de description de la page.
Rotations, translations, mises à l'échelle: tout cela, qui fait la force de PostScript, vient du fait que c'est **un langage de description de page**, et pas seulement une technique de description vectorielle des fontes. C'est pour cette raison que TrueType est bien incapable de tout cela.
C'est pour cette raison que je pense chaque jour un peu plus de bien de la Personal NT.
■



"le ToolBox-Mag américain"

Une interview exclusive de **Steven W. Disbrow**, Directeur et Rédacteur en Chef de **GS+**

ToolBox-Mag: *D'abord quelques mots, si vous le voulez bien, pour présenter à nos lecteurs l'équipe de GS+. Pouvez-vous nous dire qui vous êtes, et comment vous êtes entrés dans l'univers du GS?*

GS+: Je m'appelle ***Steven W. Disbrow***, et je suis le Rédacteur en Chef du magazine GS+. Mes premiers rapports avec l'Apple II remontent à 1984, quand j'ai acheté un IIc pour faire mes transmissions d'informatique scientifique par modem. J'ai acheté mon premier IIGS (j'en ai quatre actuellement) début 87, en vendant le IIc pour contribuer à l'achat.
Noreen Ribaric, notre Rédacteur en Chef Adjointe, est entrée dans l'univers du IIGS en entrant dans mon univers, si vous voyez ce que je veux dire... Elle fait tout le travail éditorial initial pour le magazine, et une partie de la mise en pages. Le jour, elle travaille comme programmeur pour une grosse société d'assurances. Joe et moi continuons de l'inciter à écrire un programme pour GS+, mais elle est très occupée.
Joseph Wankerl, notre Rédacteur Technique et notre as de la programmation, est entré dans l'univers du IIGS parce que sa mère a gardé le IIe quand il est parti en Faculté, et qu'il voulait le dernier cri et le meilleur de la technologie Apple II. C'est un utilisateur passionné d'Apple II depuis la classe de sixième. Joe a formé tandem avec moi après avoir trouvé le numéro 1 de GS+ dans un kiosque. Son travail est de vérifier l'exactitude technique de tout ce qui paraît dans GS+, de préparer le disque d'accompagnement du magazine, et d'écrire des programmes époustouflants. Joe est très bon sur ce point.
Suzanne Thoeming, notre Directeur de la Production, a rejoint GS+ par l'intermédiaire de Joe. Son travail est d'assurer que Joe et moi avons tout fini dans les délais (Suzanne tarabustait déjà Joe depuis deux ans, ce n'était pas un problème pour elle de tarabuster d'autres personnes tout en se faisant payer pour cela).
L'addition la plus récente à la famille GS+ est ***Wilma Tucker***. J'ai travaillé avec Wilma dans une librairie il y a deux ans. Retombant récemment sur elle par hasard au cinéma, et me rappelant combien elle était organisée et méticuleuse à la librairie, je lui ai demandé si elle aimerait faire un travail à temps partiel pour nous. Le travail de Wilma est de s'occuper de toutes ces petites choses pour lesquelles je n'ai tout simplement pas le temps: par exemple, le classement de tous les Bulletins que nous recevons de notre Connexion avec les Groupes d'Utilisateurs.

ToolBox-Mag: *Comment GS+ est-il né, et comment se développe-t-il? Pour qui veut s'enrichir, un magazine Apple IIGS est plutôt une mauvaise idée, ces temps-ci: faut-il vous ranger parmi les gens un peu fous, parmi les amoureux du GS, ou... ?*

GS+: C'est une longue histoire, mais la voici. Je travaillais comme programmeur pour le gouvernement américain sur du logiciel pour IBM, gros systèmes et Macintosh. Avoir un IIGS chez moi me permettait de rentrer à la maison sans me retrouver nez à nez avec du travail. Noreen, qui avait un travail du même genre, avait un Atari ST à la maison. Une des choses les plus sympathiques sur ST était le genre de magazines qu'on trouvait pour cette machine: des magazines avec une disquette!
J'ai pensé que ce serait chose plaisante à avoir pour le GS; j'ai donc contacté les rédacteurs de divers magazines Apple II (inCider, Nibble, A+, Call-A.P.P.L.E, etc.), pour leur demander s'ils souhaiteraient se lancer dans la publication d'un magazine de ce genre. Je n'ai jamais rencontré, sans exception, que sourires ou silence. Du coup, après avoir vainement retourné la chose dans tous les sens pendant un an (j'avais **vraiment** un bon travail, et ne voyais pas de raison de le quitter de sitôt — il me fallait en outre rassembler quelque argent pour faire sortir de terre le projet), j'ai finalement décidé de tenter de publier moi-même un magazine de ce genre. Pendant la phase préparatoire, les gens me disaient "Tu ne peux pas faire ça avec un ▶

IIGS. Achète un Mac et évite-toi les ennuis". Bien entendu, la plupart de ces gens étaient des utilisateurs de Macintosh.

Du coup, j'ai décidé que le magazine GS+ serait **entièrement** réalisé sur le GS: pas de Macs ni d'autres ordinateurs dans le coup. Noreen et moi avons mis en page le premier numéro (avec AppleWorks GS v1.0) en Septembre 89, et imprimé 10 copies sur la LaserWriter. Nous avons vendu ces copies dans les environs, et placé des publicités sur America On Line (un service d'information électronique). Commandes et argent ont commencé à arriver, en même temps que j'ai eu la chance de me voir proposer un congé à mon travail (plutôt que de le quitter purement et simplement). J'ai saisi l'occasion, et je gagne ma vie avec GS+ depuis.

Depuis ce temps, nous sommes passés de 32 pages par numéro à 60 pages (dont seulement 5 pages de publicité), et nous avons des abonnés dans le monde entier. Nous avons des rubriques sur deux services télématiques nationaux, et un serveur télématique sur lequel on peut se connecter pour poser des questions ou télécharger du logiciel.

Quant à devenir riche, eh bien, ce serait une bonne chose, et je mentirais si je disais que je ne fais pas cela pour l'argent. Mais mon but principal était seulement d'être mon propre patron, et de faire la preuve qu'on **peut** faire tout ce genre de choses avec un IIGS. Pour être productif, ni un Mac ni toute autre sorte d'ordinateur ne sont **nécessaires.**

ToolBox-Mag: GS+ en est à son douzième numéro. Quelles sont, à votre avis, les meilleures contributions publiées jusqu'ici dans votre magazine?

GS+: Hum, c'est une question dont Joe et moi avons déjà discuté, et honnêtement, nous demander de sélectionner un seul programme, c'est un peu une colle. Nous pensons sincèrement que tout programme que nous publions est un programme de valeur. Les plus **populaires** de nos programmes sont **EGOed** (un NDA éditeur de texte que j'ai écrit) et **Transfusion** (un NDA de Télécommunications écrit par Joe). Personnellement, je mets parmi mes favoris les programmes que Joe et moi avons écrits en collaboration: l'init "Défausse" *(Shuffle)* qui envoie la fenêtre de premier plan derrière toutes les autres est un de mes préférés. Je suis arrivé avec l'idée, je l'ai donnée à Joe et, une heure après, nous avions un programme qui tournait. C'est un programme simple, mais **extrêmement** utile. C'est le genre de choses que nous aimons vraiment faire.

Pour les articles, eh bien je pense qu'il faut dire que les meilleures choses que nous faisons sont les **revues de produits**. Nous insistons toujours pour des revues **détaillées**, et nous n'avons pas peur, s'agissant de programmes, de signaler les canards boîteux.

Mon exemple favori en est la revue que j'ai faite d'HyperStudio v1.0 dans notre numéro 1. L'HyperStudio original était complètement vérolé: je ne pouvais pas travailler avec plus de cinq ou dix minutes sans plantage! Un autre magazine (que je ne nommerai pas, suivez seulement mon regard) lui avait décerné la mention "Le Choix de la Rédaction" *(Editor's Choice)*. Nous avons dit aux gens de le fuir comme la peste. Je reçois encore aujourd'hui lettres et coups de fil de gens d'accord avec nous, et je suis **toujours** vu d'un mauvais œil chez Roger Wagner. Pourtant, quand HyperStudio v2.0 est sorti, nous en avons fait à nouveau la revue et, étant donné qu'ils avaient réparé tous les bugs, nous l'avons vivement recommandé à l'achat. C'est un programme vraiment bien maintenant, mais cette première version était un gaspillage de disquettes.

ToolBox-Mag: En France, quiconque veut agir pour l'Apple IIGS est contraint de se tenir soigneusement à l'écart d'Apple France. Mettre le mot "Apple" dans le titre de notre publication entraînerait à coup sûr la perte de nombreux lecteurs. La situation est-elle aussi mauvaise aux USA? Avez-vous des relations avec Apple-USA, et si oui, sont-elles bonnes?

GS+: Eh bien, disons que les choses ne vont pas **aussi** mal aux USA. Apple y a cependant une réputation solidement assise: celle de se moquer comme de l'an quarante de l'Apple II et des gens qui travaillent pour l'Apple II. Je ne reçois que par raccroc une partie du courrier adressé à Apple par des gens furieux proférant des menaces de dommages corporels envers la Direction d'Apple, mais je me demande si certains ne passent pas aux actes.

Il est cependant possible que les choses soient en train de changer. Nous avons eu un petit conflit avec Apple à propos de notre nom ("GS+"). En gros, ils nous disaient de ne pas utiliser le nom. J'ai écrit une lettre au PDG d'Apple, John Sculley. Mr Sculley s'en est occupé personnellement, et en quelques jours l'affaire a été liquidée, nous avons pu continuer à utiliser ce nom. Selon moi, c'est une preuve positive que l'attitude d'Apple envers l'Apple II est en train de changer en mieux.

ToolBox-Mag: Dans quel sens envisagez-vous le développement de GS+? L'existence de ce magazine semble être un secret pour les utilisateurs de GS: pouvez-vous y faire quelque chose? Vous concevez-vous comme une publication internationale, et les abonnements de France sont-ils les bienvenus?

GS+: L'avenir de GS+ est un sujet permanent de réflexion pour moi. D'un côté, j'aimerais répondre aux souhaits de mes lecteurs, mais de l'autre côté, il y a certaines choses que je ne ferai tout simplement pas tant que je ne serai pas certain à 100% que nous pouvons les faire bien. GS+ est un magazine qui continuera à changer, mais ce sera à petits pas.

Par exemple, nous n'avons ni couleurs ni belles images dans GS+ (en fait, nous avons une couverture en deux couleurs; mais c'est là toute l'étendue de la couleur dans GS+). Outre le fait que la couleur nous coûterait les yeux de la tête, notre objectif principal a toujours été et sera toujours de fournir de l'**information**. Si vous voulez de belles images qui mangent une demi-page, ne lisez pas GS+. Beaucoup de gens souhaitaient que nous nous mettions au papier glacé et aux photos rutilantes, mais je ne le ferai tout simplement pas pour l'instant. Je suis de ceux qui croient fermement que, s'agissant de produits informatiques, un bon dessin vaut... pratiquement rien, même pas cent mots. La seule façon de communiquer les caractéristiques d'un matériel ou d'un logiciel est de s'en servir à fond, puis d'écrire à son propos en détail.

Pour le caractère "secret" de GS+, je subodore que c'est de ma faute. J'ai toujours été si occupé rien qu'à sortir le magazine que je n'ai pas eu beaucoup d'occasions de courir pour la publicité. Mais cela va bientôt changer. Nous prévoyons de commencer à faire de la publicité dans certains bulletins de groupes d'utilisateurs américains et, si nous arrivons un jour à avoir assez d'argent, d'ouvrir une ligne à Numéro Vert et de faire de la publicité très tard le soir à la télévision. Mais il reste encore un bon bout de chemin avant cela.

Nous avons, d'ores et déjà, des abonnés du monde entier: Australie, Allemagne, France, Suède, Canada, Tahiti et Taïwan, pour n'en citer que quelques-uns. Tout abonné de tout endroit de la planète est le bienvenu! Je souhaiterais seulement que nous n'ayons pas à faire payer si cher pour un abonnement par avion en-dehors des USA. Malheureusement, il n'y a rien que nous puissions faire à ce sujet.

ToolBox-Mag: Auriez-vous quelque chose à ajouter pour les lecteurs de ToolBox-Mag?

GS+: Seulement "Merci!". J'ai réellement apprécié cette occasion de parler à vos lecteurs et de les entretenir de notre magazine GS+.

GS+, c/o EGO Systems,
P.O. Box 15366,
Chattanooga, TN 374415 - 0366, USA.

(Réalisation et traduction de l'entretien: J.Y. Bourdin)

Wanted

Depuis que nous avons la note technique Apple FTN.50.8010, nous connaissons le format des fichiers traitement de textes Appleworks-GS. Il existe un format assez universel pour les fichiers traitement de textes, appelé 'Format Texte Enrichi' (DCA-RFT). Ce format est au traitement de textes ce que le format DIF est aux tableurs: un moyen de convertir les fichiers entre programmes et entre machines différentes, en conservant le maximum possible d'information (gras, italique, mise en pages, etc). Quelqu'un aurait-il vu traîner près d'une brouette une documentation précise sur ce format? Quelqu'un aurait-il le courage de faire un utilitaire de conversion Appleworks-GS / DCA-RFT, dans les deux sens (parce que, si on compte sur Claris...)?

Actualiser Basic System

Voici un patch de ***Michaël Guitton*** au fichier `BASIC.SYSTEM` du système 5.04, pour remplacer, dans un '`CATALOG`', les types obsolètes (?) `PAS,CMD,INT,IVR` & `REL` par les types plus actuels `S16,EXE,PNT,PIC` & `SRC`.

A l'aide d'un éditeur de blocs, remplacer la séquence hexadécimale:
`FE FD FC FB FA F0 0F 06 04 EF` par: **`B0`** `FD FC` **`C1 C0 B5`** `0F 06 04` **`B3`**

Ensuite rechercher la chaîne: `PASTXTBINCMDINTIVRBASVARREL` et faire les changements suivants (***en ascii haut !***): **`S16`**`TXTBIN`**`EXEPNTPIC`**`BASVAR`**`SRC`**

Voilà c'est tout!

Une méthode pour appeler des
routines externes en TML PASCAL

Stéphane Hadinger

Les Compilateurs TML PASCAL I & II sont rapides, simples à utiliser et performants, mais ils ont un défaut de taille, ils sont fermés. Il est impossible d'utiliser des routines en assembleur ou en C dans un programme en Pascal. En fait, s'il est impossible d'inclure des routines externes par les voies normales, il existe des moyens détournés.
Le premier consiste à créer un nouvel outil et à y inclure ses routines externes, puis à définir les nouveaux appels d'outil en Unit. Ce moyen est simple, universel mais assez lourd à mettre en œuvre puisque l'outil doit se trouver en permanence dans le dossier /SYSTEM/TOOLS de votre disque système. Ce n'est utilisable que dans le cas de routines d'usage général, pouvant servir dans diverses applications, comme par exemple ***Music Juke-Box*** (cf. ToolBox-Mag n°1).
Cependant si vous voulez accélérer telle ou telle portion de votre application, il serait peu judicieux d'inclure les routines en assembleur dans un outil, ce qui aboutirait à avoir un outil par application. Cela va à l'encontre de la philosophie Apple.

Je vous propose ici une alternative, une méthode notamment utilisée dans ***Fontasm***: elle consiste à **regrouper toutes les routines externes dans un fichier exécutable auxiliaire, à charger ce fichier en mémoire à partir de votre application en Pascal et à appeler ces routines directement en mémoire**.

Mise en place du fichier auxiliaire

Vous devez tout d'abord regrouper toutes vos routines dans un unique fichier, dont l'en-tête doit avoir certaines particularités. Sous APW créez le fichier suivant (*ExternAsm.Src*):

```
EXTERN  START
        RTL
        JMP    ROUTINE1
        JMP    ROUTINE2
        .......
        END
```

Assemblez-le (*ASSEMBLE EXTERNASM KEEP=EXTERNASM*) et linkez le tout en faisant bien attention de placer *EXTERNASM.ROOT* au début du segment principal. Par exemple tapez *LINK EXTERNASM ROUTINES KEEP=ROUTINES.*

Le type de fichier sera au choix EXE ou RTL. Le type RTL (Run Time Library) convient bien car on le charge par le System Loader, et il n'a jamais été et ne sera probablement jamais utilisé. Vous pouvez aussi utiliser le type EXE qui est le type par défaut. L'instruction RTL située en début de fichier permet une sortie propre en cas d'exécution accidentelle du fichier auxiliaire, et est aussi indispensable car la table des JMP doit commencer au deuxième octet du fichier auxiliaire. Si on la faisait commencer au premier octet, cette table risquerait d'être chargée au début d'un banc mémoire, ce qui poserait des problèmes.

L'interface pour TML PASCAL.

Vous trouverez sur la disquette les sources à utiliser. L'Unit *Extern.Pas* (cf. Listing 1) contient tout ce qui est nécessaire pour l'appel de vos routines. Compilez-la puis créez une nouvelle Unit contenant les appels à vos routines sur le modèle du listing 2.

Maintenant vient la partie la plus difficile du travail, il faut **coder manuellement les échanges de paramètres entre vos routines et l'application en Pascal**. Il est évident que vos routines externes doivent suivre les standards d'appels de routines Pascal en ce qui concerne la pile, c'est-à-dire que les paramètres doivent être passés via la pile.

Les routines PushInt, PushLong et PushPtr permettent d'empiler les arguments de types Integer, LongInt et Ptr (pointer). L'empilement des paramètres se fait dans l'ordre d'apparition des arguments dans la définition de la routine en Pascal. Notez bien que si un argument est passé par variable (Var) et non par valeur, c'est l'adresse de l'argument qui est empilée et non pas sa valeur (cf. Listing 4, Routine3). De ▶

même si un argument dépasse quatre octets, il est copié dans un coin de mémoire par le compilateur Pascal et c'est l'adresse de la copie qui est transmise, dans ce cas il faut aussi empiler l'adresse de l'argument (cf. Listing 4, Routine5). Dans le cas d'une fonction, n'oubliez pas d'empiler, avant les paramètres, des octets nuls en nombre suffisant pour laisser la place sur la pile pour le résultat. Ainsi pour une fonction entière faites PushInt(0), pour une fonction renvoyant un entier long faites PushLong(0) (cf. Listing 4). Après l'appel de la fonction, récupérez le résultat en le dépilant. PullInt permet de dépiler un entier. Il n'existe pas de PullLong ni de PullPtr, référez-vous aux exemples du listing 4 pour voir comment procéder. Suivez bien ces exemples de routines fictives et ne tentez pas d'écrire votre propre fonction PullLong, cela ne marcherait pas.

La méthode proposée ici repose entièrement sur du bidouillage au niveau assembleur et pile à partir d'un programme en Pascal. Elle ne supporte donc pas d'originalité de votre part, et les méthodes décrites ici doivent être suivies à la lettre sous peine de joyeux plantages.

L'appel effectif de la routine se fait par la séquence :

```
js1; js2; js3; js4;
PushLong(RoutineX);
js5; js6; js7; js8; js9; jsA; jsB; jsC; jsD; jsE;
```

Respectez bien cette syntaxe car tout repose sur le nombre d'octets qu'elle occupe une fois compilée. Il ne doit pas être fait de calcul dans le PushLong. De plus il est important que RoutineX soit une variable LongInt *globale* (pas locale à la procédure) située dans le *segment de données principal* (s'il n'y en n'a qu'un seul alors tout va bien). Le listing 4 donne plusieurs exemples en faisant le tour des principaux cas de figure envisageables.

Voilà, compilez votre application en n'oubliant pas les *USES Extern, MesRoutines;* et l'appel à *ChargeRoutines;* en début de programme. ■

Le GS et le système 7 du Mac

J.Y. Bourdin

Sur GS, l'arrivée d'un nouveau système d'Apple signifie jusqu'ici une nouvelle et considérable *accélération* du fonctionnement du système (songez au 4.0, puis au 5.0). Sur Mac, ça n'est pas le cas: le système 7.0 fait payer bien cher aux utilisateurs de Mac les sous-catalogues dans le catalogue Système et ses autres innovations. Si cela est indifférent aux utilisateurs de GS, en revanche deux points méritent leur attention dans le nouveau système du Mac:

• D'une part, il suffit maintenant d'un câble AppleTalk pour envoyer et recevoir, depuis le GS, des fichiers sur le disque dur du Mac. La supériorité par rapport à Appleshare (outre le coût), c'est que le Mac n'a plus besoin d'être dédié en serveur de fichiers, il peut fonctionner en même temps. On peut même, sur le GS, lancer HyperCard GS depuis AppleTalk, c'est-à-dire depuis le disque dur du Mac. Il faut pas mal de patience, mais c'est le seul moyen pour un utilisateur de Mac d'avoir HyperCard en couleurs sur son disque dur.

• D'autre part, Apple File Exchange a enfin appris qu'on pouvait mettre du logiciel HFS/Mac sur un amovible Syquest, il ne s'affole plus quand il en rencontre. Il aura fallu le temps…
Cela ne suffit pourtant pas pour qu'on puisse conseiller l'usage d'Apple File Exchange: si vous avez *deux* Syquest chaînés, il reconnaîtra une seconde cartouche HFS/Mac (à condition toutefois que vous "montiez" le volume correspondant avec l'utilitaire adéquat). Mais si, sous Finder, vous éjectez cette seconde cartouche, laissez le Syquest correspondant vide, et lancez *ensuite* AFE, alors là, gare à votre Syquest! *Eteignez le plus vite possible!*
Si vous pensez par ailleurs qu'Apple File Exchange va reconnaître une cartouche Syquest formatée en GS/OS, vous croyez au Père Noël: il ne reconnaît rien. Il ne sait toujours pas non plus transporter correctement les fichiers sparse de Prodos, ni les fichiers GS avec ressources. Bref, AFE reste insuffisant, bugué et dangereux. Si vous avez des conversions Mac/GS à faire, faites le maximum sur le GS, c'est plus sûr.
Si vous décidez de négliger mon conseil et de vous servir quand même d'AFE, alors j'ai un second conseil: vous diminuerez les risques en utilisant AFE sous le système 6.07 du Mac plutôt que sous système 7. Expliquer pourquoi nous ferait sortir du cadre de ToolBox-Mag: il suffira de noter qu'une des disquettes du système 7 (la disquette "Utilitaires 2") contient, heureux hasard, un système… 6.07 qui boote et tourne sur tous les Macs.

Gate:

un jeu-marathon

Laurent Bourdin

Après des dizaines d'heures d'essais, au bout d'une partie de quatre heures consécutives, ça y est: **j'ai fini Gate**! Enfin l'horrible Darg est vaincu.

Gate, le jeu-marathon de Bright Software & ToolBox, écrit par *Henrik Gudat, Joerg Kienzle et Yann Le Tensorer*, présente une multiplicité de difficultés et suscite une passion dévorante. C'est un jeu de rôle et d'action dans le genre Ultima ou Sword of Kadash. Mais il est moins compliqué qu'Ultima V, on peut y jouer sans ouvrir une Base de Données.

La documentation vous suppose déjà initié aux jeux de rôles: sans quoi, elle est peut-être un peu succincte. Il faut tuer tous les monstres qui protègent Darg, augmenter la richesse de votre personnage (il existe des boutiques sur Divesia, elles vendent armes, armures et remèdes pour retrouver la santé), sa force et ses armes pour mieux combattre au niveau suivant.
Il faut aussi tout fouiller, ramasser des parchemins; creuser pour les trouver (certains, au moins) ou pour découvrir des téléporteurs, des clés, etc; répondre à des questions à partir d'indications d'autant plus imprécises que l'on avance dans le jeu.

On joue à Gate armé de feuilles de papier pour noter les messages des parchemins, et de papier quadrillé pour s'y retrouver (avec les téléporteurs, le labyrinthe n'est pas simple…). Le jeu se manipule au clavier ou au joystick, mais je recommande très vivement le joystick (au clavier, le personnage ne s'arrête jamais, il peut seulement repartir dans une autre direction).

Gate est un jeu passionnant et prenant, mais difficile. Heureusement, on a la possibilité de sauver la partie en cours, cela utilisera une disquette pour chaque sauvegarde.
Le principal défaut de Gate est d'ailleurs l'impossibilité de l'installer sur disque dur, car ce jeu, livré sur deux disquettes, est protégé. Du coup, le chargement est un peu lent et, dans une configuration avec un seul lecteur 3,5 plus disque dur, il faut trop souvent changer de disquette: il faut également avoir sous la main une provision de disquettes formatées pour sauver les parties en cours.

Globalement, Gate semble un peu ingrat au début: les plus beaux graphismes se trouvent dans les niveaux élevés. Après avoir perdu (ce qui arrive souvent au début), il faut tout reconfigurer, le langage d'utilisation (car Gate est bilingue français/anglais, ainsi que sa documentation), comme le joystick, car on recommence le jeu en rebootant.

Soyez donc courageux au début, car la passion est un peu lente à venir; mais quand elle y est, quand vous avez passé le niveau 3, c'est de la **rage**. Il faut **absolument** aller jusqu'au bout!
Avec sa musique stéréo, son tableau de bord complet, Gate est un vrai jeu GS, exploitant les possibilités graphiques et sonores de la machine. Je ne peux que le recommander. ■

Les pièges de Gate

• Possesseurs de RamKeeper, attention! Avec deux lecteurs 3 1/2 et RamKeeper, je n'ai pas réussi à booter Gate. Avec un seul lecteur, j'ai réussi à booter, charger un jeu sauvegardé, jouer. Mais, quand j'ai voulu sauver le jeu, j'ai écrasé le contenu de ma RamKeeper.
• Si, après avoir creusé à un endroit, on découvre un téléporteur, ou un éclair, ou un message, et qu'après cette découverte, pour une raison ou une autre, on charge une sauvegarde en passant par la fonction 'Charger' du tableau de bord, en creusant à nouveau ensuite au même endroit, *on ne trouve plus rien*. Pour contrer cette ruse de l'ignoble Darg, un moyen: *rebooter la disquette*.

JLT

Gate: conseils pour sauver Divésia

Jean-Louis Torre

Divesia est pillée, ravagée par les hordes de Darg. Fier héros, tu as déjà entamé la lutte contre le Roi félon. Mais, faute d'or, tu n'as pas pu t'acheter une bonne armure ou des éclairs supplémentaires pour te battre dans des conditions décentes. Tu croupis au deuxième niveau, et le calvaire de Divésia se poursuit.

Reprends les armes chevalier, la lutte continue! Le Conseil des Sages de Divésia, devant les horreurs perpétrées par Darg, a décidé d'intervenir et de donner quelques conseils aux preux qui n'ont pas encore terminé leur mission. Avant de reprendre la route, pénètre-toi donc de nos conseils:

• Le but de ta quête est de trouver Darg pour le vaincre en duel. Mais ne sois pas impatient: auparavant, il te faudra tout fouiller, détruire tous les monstres que tu rencontreras et ramasser tous les objets que tu trouveras. Trois de ces objets te seront indispensables pour ton duel final avec le Roi félon.

• Il ne te suffira pas de détruire les ignobles monstres qui protègent Darg. Il faudra surtout détruire leurs tanières, ces grands X rouges ou bleus que sont les nids qui les engendrent en permanence.

• Tu auras souvent à creuser le sol de la forteresse de Darg, grâce à la touche Annulation (celle qui surplombe le 7 du pavé numérique). Pour cesser de creuser, tu devras taper une autre touche: résiste à la tentation d'utiliser la barre d'espace, car elle déchaîne des éclairs qui te sont fort précieux.

• Ces éclairs de la barre d'espace te seront très utiles: ils détruisent tous les monstres, aussi bien que les ignobles 'nids' de ces monstres que tu découvriras. La foudre sera même ta seule arme contre certains gros monstres velus.

• Pour certaines armes, n'oublie pas que tu peux changer de style de tir en tapant Ctrl-*.

• Au début, tu ne pourras porter que trois objets (éclairs, clefs rouges, clefs jaunes). Pour augmenter ce nombre magique “Max”, ramasse les bouteilles bleues, et achète dans les boutiques.

• Sache que l'ignoble Darg a imposé le monopole dans les boutiques: d'une échoppe à l'autre, les prix sont toujours les mêmes.

• Sache aussi que dans les boutiques, les éclairs ne sont pas toujours des éclairs.

• Certains objets (anneaux, bouteilles ou éclairs) jouent le rôle d'une barrière magique contre les monstres: ils interdisent leur progression, mais pas la tienne. Tu pourras aussi tirer sans risque contre les monstres à travers les barrières invisibles qu'ils constituent.

• Pour te repérer dans la labyrinthe démoniaque de Darg, songe au Petit Poucet: si tu n'as pas de cailloux, tu pourras néanmoins déposer des objets, clefs ou éclairs, pour te repérer et te réarmer en cas de besoin. La touche Del te le permettra.

• Parfois te seront posées certaines énigmes fort cruelles. Souviens-toi donc de cette sentence des Anciens de Divésia, qui t'aidera dans ta quête: *En 64 semaines de temps, passez le corridor et le pont bleu, pour rejoindre le trou noir des illusions magiques.*

Allons, chevalier, l'heure est venue pour toi d'affronter ton destin. Arme-toi de courage et de patience, et sauve Divésia! ■

Introduction à la programmation de l'Apple II GS

Du Basic au Pascal, 3º partie : Le déroulement des instructions

Jean Destelle

Après avoir passé en revue les différentes déclarations obligatoires du Pascal, nous allons maintenant montrer, par des exemples, l'emploi des diverses instructions du Pascal dans le déroulement du programme. Les programmes et sources auxquels cet article se réfère sont sur les disquettes ToolBox-Mag 5 et 6.

Dans le langage Pascal, on distingue les *instructions simples* et les *instructions structurées*.

1. Les instructions simples

Ce sont celles qui ne comprennent aucune autre instruction: les assignations, les instructions de procédure, les branchements.

Les assignations:

- Un nombre: *LaLongueur := 14;* le mot-clé := indique l'assignation. On assigne la valeur écrite du côté droit à la variable écrite à gauche.
- Une chaîne: *MaChaine := ' Monsieur Pascal';*
- Un élément de Tableau: *Age[4] := 17;* le nombre 4 est l'Index, le numéro d'ordre de l'élément assigné.
- Le champ d'un record: *MonIOParamsRec.pcount := 4;* le champ *pCount* du record *MonIOParamsRec* prend la valeur 4.
- Une variable dynamique: *laGSstring := @LaChaine;*
- Autres exemples d'assignations: *a := a+4; x := (3* sqr (y) - (b div c));*

Les instructions de procédure:

Exemples: *DoClose; SaveMyFile (laFile, lepathname); Moveto (10,20);* ces instructions constituent un appel à la procédure désignée par son identificateur *(DoClose, SaveMyFile* ou *MoveTo).* Entre parenthèses figurent les paramètres, obligatoirement dans l'ordre indiqué lors de la déclaration de la procédure.

Les branchements:

Exemple: *if a >= 10 then GoTo 20;* le nombre 20 est une étiquette *(label)*, qui a dû être déclarée comme telle précédemment. *GoTo* renvoie au point d'entrée, qui est précédé par l'étiquette suivie de ':'.
Contrairement au Basic et à l'assembleur, ce type de branchement est peu utilisé en Pascal, puisque la plupart des branchements se font au moyen d'appels de fonctions ou de procédures.

2. Les instructions structurées

Ce sont celles qui se composent de plusieurs instructions successives: les instructions composées, les instructions conditionnelles, les instructions de répétition.

2.1. Les instructions composées

Elles sont formées de plusieurs instructions successives, formant une suite débutant par *begin* et se terminant par *end*. Un tel bloc est traité comme si c'était une seule instruction. Un exemple pris dans le source de *AuxSelect1*, dans la procédure *StartUpGSTools*:

```
if _ToolErr > noError then
  begin
  e:= _ToolErr;
  Writeln('Erreur en chargeant un outil...Erreur #
',e);
  Halt;
  end;
```

2.2. Les instructions conditionnelles

Deux moyens sont disponibles dans le Pascal: *if ... then ... else ...* et *case of ... otherwise ...*

2.2.1. L'instruction if

Cette instruction s'emploie pratiquement comme en Basic. Seul le mot-clé *else* peut être ajouté, ce qui permet de faciliter parfois l'écriture. Nous prendrons comme exemple la procédure *DoClose* qui ferme la fenêtre supérieure, celle qui est active, et peut être connue en utilisant la fonction *FrontWindow* de WindowMgr:

```
procedure DoClose; (*fermer la fenêtre supérieure*)
var lafenetre: WindowPtr;
begin
  lafenetre := FrontWindow;
  if lafenetre <> nil then (*la fenêtre existe*)
    if GetWkind(lafenetre) = 0 (*si c'est une fenêtre
d'application*)
        then CloseWindow(lafenetre) (*on la ferme*)
    else CloseNDAByWinPtr(lafenetre); (*fermer la
fenêtre NDA*)
end; (*do DoClose*)
```

Dans cet exemple, figurent deux instructions 

conditionnelles imbriquées. La première, *if lafenetre <> nil then*, ne comporte pas de *else*. La seconde, *if GetWKind(lafenetre) = 0 then CloseWindow(lafenetre)*, est suivie d'un *else: else CloseNDAByWinPtr(lafenetre);*

A la suite de *if* doit figurer une expression ou une variable booléenne (comme en Basic). Si sa valeur est "vrai" *(true)*, l'instruction (ou le bloc d'instructions) qui suit est exécutée. Dans le cas contraire, le programme saute à ce qui suit le premier *else* (un *else* se rapporte toujours au dernier *if*). S'il n'y a pas de *else*, le programme continue par l'instruction suivante. Notez qu'il n'y a pas de point-virgule avant *else*.

Vous trouverez de nombreux exemples d'emploi de l'instruction *if ... then ... else* dans les sources Pascal à votre disposition. Vous noterez que, souvent, le mot *else* ne sert à rien. Il est placé pour rendre la lecture plus claire, mais pourrait être remplacé par ';'. Dans d'autre cas, sa présence est indispensable. Ne supprimez pas d'office un *else* qui vous semble inutile: il peut avoir un emploi peu évident, comme tout simplement de permettre au *else* qui suit de se rapporter au bon *if*.

A propos du Window Manager

Profitons de l'exemple de cette fonction *DoClose* pour commenter l'emploi de quelques fonctions de WindowMgr. Cette procédure est des plus classiques et vous pouvez la considérer comme standard. Bien que très simple, elle répond parfaitement au but: fermer la fenêtre du dessus.

La fonction *FrontWindow* retourne un *windowptr* sur la fenêtre sélectionnée; plus précisément sur le *WindowRecord*, une structure comportant tous les renseignements, actualisés en permanence, sur cette fenêtre. Le type *windowptr* est défini comme un *GrafPortptr*, pointeur vers un *grafport*. Le premier champ du *windowRecord* étant le *grafport* de la fenêtre, le *windowptr* pointe donc aussi sur le *Windowrec*.

S'il n'y a pas de fenêtre ouverte et sélectionnée, *FrontWindow* retourne *Nil*. Dans ces conditions, le programme saute à *end*, et la procédure est terminée. S'il y a une fenêtre ouverte, on examine quel est son type: soit une fenêtre de l'application, soit une fenêtre du système (un NDA). La fonction *GetWkind* retourne zéro dans le premier cas. On peut alors employer la procédure *CloseWindow* qui admet comme paramètre le *windowptr* de la fenêtre, pour fermer complètement la fenêtre et disposer de tous ses éléments. La fenêtre n'existe plus alors.

Si la valeur retournée par *GetWkind* est différente de zéro, il s'agit d'une fenêtre-système. On appelle alors une procédure spéciale (et nouvelle) de l'outil DeskMgr: *CloseNDAByWinPtr* avec, comme paramètre, le *windowptr*. Cette procédure ferme le NDA *(ce qui n'est pas un mince travail. Bravo Apple!)*.

2.2.2 L'instruction Case

Le mot-clé *Case* est suivi d'un *sélecteur* qui est une *expression*, c'est-à-dire de l'identificateur d'une variable ou d'une expression mathématique comportant une ou plusieurs variables, puis du mot-clé *of*. Ensuite, viennent les différents "cas" examinés; puis la clause éventuelle *otherwise*, qui est suivie d'une instruction ou d'un bloc d'instructions. L'instruction se termine enfin par *end*. Nous prendrons un exemple extrait de la procédure *MainEventLoop* de *MiniDeskTop:*

```
code := TaskMaster($FFFF, gmainEvent);
(*TaskMaster recueille l'événement*)
  case code of (*il examine les cas intéressants*)
    winGoaway: doClose; (*fermeture de fenêtre*)
    winspecial, winMenuBar: GererMenu; (*clic
dans un item de menu*)
    wincontrol: GererControles; (*on a agi dans un
contrôle*)
  end; (*de case*)
```

A chaque tour de la boucle d'événements, le programme fait appel à TaskMaster, au moyen de la fonction *TaskMaster*, qui retourne un integer désigné ici par la variable *code*. Nous devons ensuite comparer la valeur de *code* à différentes valeurs possibles, qui sont ici désignées par des "constantes" normalisées par Apple, et dont les valeurs ont été déclarées dans l'unit d'interface "Windows". Il y a un grand nombre de cas possibles. Dans le programme *MiniDeskTop*, qui ne fait que peu de choses, nous nous contentons seulement des quatre valeurs qui nous intéressent: *wingoaway* (qui vaut $0016) indiquera qu'on a cliqué dans la case de fermeture de la fenêtre; *winspecial*, *winMenuBar* ($0019 et $0011) nous indiquent qu'il s'agit de menus; *wincontrol* ($0021) indique qu'on a agi dans un contrôle.

Dans la ligne *case code of*, la variable *code* est le sélecteur. Chaque cas examiné ensuite donne lieu à une instruction du genre: *winGoaway: doClose; winGoaway* est la constante à laquelle on compare *code*. Notez qu'on aurait pu tout aussi bien écrire la valeur numérique. Après les "deux-points", se trouve l'instruction ou le bloc d'instructions à déclencher dans ce cas. Si on ne fait rien, on ne met qu'un point-virgule. Dans cet exemple, on appelle la procédure *DoClose*, qui ferme la fenêtre, et que nous avons étudiée ci-dessus.

Dans cet exemple, il n'y a pas de clause *otherwise* ("autrement") qui serait suivie d'une instruction ou d'un bloc d'instructions à accomplir pour toute autre valeur de *code*. Vous comprenez aisément pourquoi on n'en met pas. Il y a tellement de cas possibles qu'on ne peut pas indiquer un processus applicable à l'ensemble des cas: mieux vaut ne rien faire, et laisser TaskMaster s'en charger.

La ligne *winspecial, winMenuBar: GererMenu;* vous montre comment on peut indiquer plusieurs constantes successives, séparées par des virgules, qui provoqueront la même action. Enfin, le ▶

mot *end;* indique au compilateur que la série des comparaisons est terminée.
Notez que nous pourrons, quand nous compléterons ce programme, rajouter d'autres cas dans la liste des comparaisons. Vous trouverez très souvent ce genre d'instruction dans nos programmes. C'est très pratique, et très clair à relire. Voyez par exemple dans la procédure *GérerMenu,* qui examine le numéro de l'item de menu détecté, et enclenche, pour chaque cas, les instructions voulues.

2.3. Les Instructions de répétition

Elles servent à construire des "boucles" dans le programme, afin de repasser plusieurs fois sur les mêmes instructions. Le Pascal est ici nettement plus riche que le Basic. Il nous offre trois procédés différents: les instructions *for ... to ... do, repeat ... until,* et *While ... do.*

2.3.1 L'instruction *for*

Elle est très similaire à celle du Basic, mais ne comporte pas de *Next* en fin de boucle. Le caractère très général du Pascal rend l'emploi de cette instruction possible dans de nombreux cas. Nous ne considérerons que son emploi le plus général, qui consiste à utiliser un nombre entier pour la "variable de contrôle".
Prenons un exemple: *for compteur := 1 to 10 do SysBeep;* la variable *compteur,* variable de contrôle, doit avoir été déclarée dans un type compatible avec les valeurs 1 et 10 qui sont les bornes de comptage (début et fin). Il n'y a pas, comme en Basic, d'instruction *Step* pour fixer le pas d'accroissement de la variable. Cette dernière doit en effet appartenir à un type "scalaire", donc l'accroissement se fait par passage d'un élément au suivant, dans l'ordre croissant des valeurs ordinales. En bon français, cela veut dire que pour des nombres entiers le pas d'accroissement sera de 1. Si vous désirez fonctionner dans l'ordre décroissant, il vous faudra utiliser le mot-clé *downto* au lieu de *to.*
A la suite de *do* vous pouvez écrire une instruction simple, ou un bloc d'instructions encadré par *begin* et *end.* L'exemple ci-dessus provoquera l'actionnement de 10 "bips" successifs.

Voici un autre exemple, prélevé dans la fonction *BreakAlert* de *AuxSelect1:*

```
for a:= 1 to length(lemessage) do
   if lemessage[a]='/' then lemessage[a]:=':';
```

Cette ligne d'instruction sert à remplacer, dans une chaîne dont le nom de variable est *lemessage,* tous les caractères '/' par un caractère ':'. En effet, *BreakAlert* fait appel à la fonction *AlertWindow* de WindowMgr, qui donne un sens tout à fait spécial, de séparateur, au caractère '/'. De sorte qu'on ne peut pas afficher n'importe quel chemin d'accès de fichier avec cet outil. *Length(lemessage)* est le nombre de mots de la chaîne. *LeMessage[a]* est le caractère d'ordre *a* de la chaîne. Après *do,* nous n'avons pas besoin d'écrire *begin,* car l'instruction qui suit est une instruction structurée, mais pas un bloc.
En revanche, si nous voulions en plus convertir dans cette chaîne, les minuscules en majuscules nous écririons:

```
var a:integer; c: char;
  for a := 1 to length(lemessage) do begin
     c := lemessage[a];
        if c ='/' then lemessage[a]:=':';
        if c in ['a'..'z'] then lemessage[a] :=
chr(ord(c) -32);
  end;
```

A chaque tour de boucle, les instructions comprises entre *begin* et *end* sont exécutées. Si le caractère rencontré est une minuscule comprise entre 'a' et 'z', on transforme ce caractère en majuscule en soustrayant 32 à son code ASCII. Remarquez l'emploi de l'instruction *in['a'..'z']* qui se lit: *fait partie de l'ensemble composé de l'énumération des caractères de a à z.*
Il est interdit de modifier la valeur de la variable de contrôle dans les instructions à l'intérieur de la boucle. Cela est fait automatiquement lors du renvoi, par *end,* à l'instruction *for.*
La boucle est au moins parcourue une fois, même si les deux bornes sont égales. Si la borne finale est impossible à atteindre, parce que sa valeur est mal choisie, il peut se produire un "bouclage sans fin" qui bloque le programme.

2.3.2. L'instruction *Leave*

Il est possible de sortir de la boucle avant qu'elle n'ait effectué le nombre prévu de tours. Pour cela, nous devons utiliser l'instruction *Leave,* qui fait quitter la boucle et continuer par la première instruction qui suit la boucle. Par exemple:

```
for a := 1 to length(letexte) do
  if letexte[a] = 'P' then begin
     Ppresent:= true;
     leave;
  end;
```

Ces instructions recherchent la lettre 'P' dans la chaîne *letexte,* et dès qu'elle est trouvée, le rebouclage est interrompu, ce qui permettra de gagner du temps dans la recherche. Au moment de l'interruption, la valeur de *a* représente le nombre de tours effectué, et, dans notre exemple, le numéro d'ordre de la lettre trouvée.

2.3.3. L'instruction *cycle*

Il est possible aussi de "sauter", d'une instruction située à l'intérieur de la boucle, directement au début de la boucle, au moyen de l'instruction *cycle.* Par exemple, modifions encore l'exemple de la chaîne *lemessage:*

```
for a := 1 to length(lemessage) do begin
  c:= lemessage [a];
  if c ='/' then begin
```

▶

lemessage[a]:=':';
cycle;
end;
if c in ['a'..'z'] then lemessage[a] := chr(ord(c) - 32);
end;

Nous avons ajouté *cycle* après le changement de '/' en ':'. De ce fait, l'instruction suivante (qui est inutile dans ce cas) ne se fera pas.

2.3.4 L'instruction *Repeat ... until*

Ce type de boucle, inconnu du Basic, est très utilisé en Pascal. Les instructions qui suivent *repeat* sont recommencées jusqu'à ce que la condition booléenne qui suit *until* (jusqu'à) soit vraie. Voici par exemple comment nous écririons la recherche de la lettre P dans la chaîne *LeTexte*, au moyen de cette instruction:

a := 0;
repeat
inc(a);
until letexte[a] = 'P' or (a >= length(a));

Cette façon d'écrire a le mérite de la clarté. Mais attention! Il est indispensable, lorsqu'on écrit des instructions de répétition, de s'assurer que dans tous les cas possibles, la boucle s'arrêtera. C'est la raison d'être de *or (a >= length(a));* Si cette instruction n'était pas écrite, et si le texte ne comportait pas la lettre 'P', la boucle continuerait de tourner sans fin.

Remarquez aussi l'emploi de >= et non de = dans cette dernière condition. C'est une habitude à prendre. On ne sait jamais. Dans une procédure un peu compliquée, il pourrait arriver qu'on "saute" la valeur qui sert de borne. Car rien ne vous empêche, avec l'instruction *repeat*, de modifier plusieurs fois les valeurs des variables servant dans l'expression booléenne qui suit *until*.

Nous avons utilisé une fonction standard *inc(a)*, qui permet d'incrémenter (c'est à dire de majorer de 1) une variable entière. *Inc(a);* équivaut exactement à *a := a + 1;* Il existe de même, pour décrémenter, une fonction *dec(a);* équivalente à *a := a-1;*).

On peut utiliser avec repeat les instructions *Leave* et *Cycle*, de la même façon qu'avec *For*.

La Boucle d'événements de *MiniDeskTop*

Voici l'emploi le plus fondamental de la programmation du GS en "DeskTop", qui repose sur l'emploi d'une boucle d'événements, qui tourne de façon quasi permanente tant que le programme fonctionne. Lorsqu'un événement apparaît, comme un clic dans un menu ou l'actionnement d'un contrôle, la boucle s'arrête un court instant pour laisser l'action commandée se dérouler, puis reprend sa répétition, jusqu'à ce que l'un des événements provoque l'indication de la fin du programme, par la mise à *true* de la variable booléenne *done*.

procedure MainEventLoop;
var code: integer;
begin
*gMainevent.wmTaskMask := $001FFFFF; (*on laisse tout faire à TaskMaster*)*
*done:= false; (*done est toujours faux, sauf quand doQuit l'a changé*)*
*repeat (*début de la grande boucle ...*)*
*code := TaskMaster($FFFF, gmainEvent); (*taskMaster recueille l'événement*)*
*case code of (*il examine les cas intéressants ...*)*
*winGoaway: doClose; (*clic dans case de fermeture de fenêtre*)*
*winspecial, winMenuBar: GererMenu; (*clic dans un item de menu*)*
*wincontrol: GererControles; (*on a agi dans un contrôle*)*
*end; (*de case*)*
*until done; (*si done:= true, on sort ici*)*
end;

Dans cet exemple, la variable booléenne *done* sera mise à *false* quand on actionnera le choix "Quitter" du menu, qui appellera la procédure *DoQuit*, un outil standard ultra-simple de *AuxSelect1:*

procedure DoQuit;
begin
Done := true;
*end; (*de DoQuit*)*

Pour mettre fin à un programme en Desktop, il suffit donc de faire cesser le rebouclage de la boucle principale d'événements. Il y aura très souvent d'autres boucles, "imbriquées" à l'intérieur de la boucle principale, comme les boucles "modales" destinées à traiter les contrôles d'une fenêtre de dialogue. Voyez l'article sur la gestion des événements

2.3.5. L'instruction *While ... do*

While signifie "tant que". Contrairement à *repeat*, la condition booléenne est indiquée après *while*, en début de boucle. Si plusieurs instructions doivent être exécutées dans la boucle, il faut constituer un bloc en les encadrant par *begin ... end*.

Les mêmes précautions d'emploi sont nécessaires qu'avec *repeat*. La différence principale réside dans le fait que la condition de fin est examinée en début de boucle. Donc, si la condition n'est pas remplie au départ, la boucle n'est pas exécutée. (Au contraire, avec *repeat* la boucle est exécutée toujours au moins une fois). Comme pour *repeat*, il faut impérativement s'assurer que la boucle ne risque pas de tourner sans fin, en ajoutant au besoin une limite au nombre de tours. On ne sait jamais...

Reprenons l'exemple de la recherche de la lettre 'P' dans la chaîne *LeTexte*. En utilisant *while*, nous l'écrirons:

a := 0;
while (not letexte[a] = 'P') or (a < length(a)) do
inc(a);

▶

Comme la boucle tourne tant que la condition est remplie, les conditions ont dû être inversées par rapport à celles de *repeat*. Cela risque de rendre la compréhension du programme, à la lecture, moins évidente.
Pour montrer l'emploi de *begin ... end* dans une boucle utilisant *while*, nous reprendrons l'exemple de la modification de la chaîne *leMessage*, ci-dessus.

```
a:=0;
while a <= length(lemessage) do
  begin
    c := lemessage[a];
    if c ='/' then lemessage[a]:=':';
    if c in ['a'..'z'] then lemessage[a] := chr(ord(c) -
32);
  end;
```

On peut utiliser *Cycle* et *Leave* avec *while* comme avec *repeat* et *for*.

3. Procédures et fonctions

Pascal nous permet de créer des "routines" c'est-à-dire des "sous-programmes", que l'on peut appeler de tout point de notre programme, en leur transmettant des paramètres d'entrée, et qui peuvent fournir des résultats sous forme de paramètres de sortie. Deux possibilités: les procédures, et les fonctions. Les fonctions sont en fait simplement des procédures dont l'identificateur peut être utilisé comme le nom d'une autre variable, et qui représentent une valeur, comme toute variable.
Fonctions et procédures en Pascal peuvent être "récursives". C'est-à-dire qu'elles peuvent s'appeler elle-mêmes, depuis l'une des instructions qu'elles contiennent. A utiliser avec prudence, cela peut être la cause de rebouclages intempestifs.

3.1. Les procédures

L'écriture d'une procédure que la théorie du Pascal dénomme "déclaration de procédure", comporte deux parties: l'entête, qui contient la déclaration de l'identificateur et éventuellement des paramètres "formels", et le "corps" de la procédure, qui forme un "bloc" au sens où nous l'avons défini lors de l'étude des déclarations. Ce bloc est donc composé des déclarations des étiquettes, constantes et variables locales, et de la séquence des instructions, qui peut elle-même être composée d'autres blocs. Le "corps" de la procédure peut être remplacé par une "directive".

3.1.1. Les Directives

Les directives utilisées par TML Pascal II sont: *Forward, External, Inline* et *Tool*. Chacune de ces directives indique que le contenu de la procédure se trouve ailleurs:
Forward indique que la procédure est décrite plus loin dans ce programme: *procedure DessinRectangle(x:integer;y:integer); Forward;* dans ce cas, cette déclaration doit être faite dans les mêmes formes, avec les mêmes paramètres que la déclaration de procédure que l'on trouvera plus loin.
External indique que le code compilé par un autre langage, est disponible ailleurs que dans les units ou le programme en cours.
Inline est suivi directement du code-machine hexa à exécuter. On l'emploie pour insérer un ordre très bref, écrit en code-machine. Voyez l'article de Bernard Fournier dans ToolBox-Mag 5.
Tool indique que la procédure est l'un des outils de la Toolbox du GS, et est suivi de la référence de l'outil (numéro de l'outil, virgule, numéro de fonction): *procedure MoveTo (h,v: integer); Tool 4, 58;*

3.1.2. Les procédures

Voici quelques exemples de procédures:
• une procédure de *AuxSelect1*, qui crée les couleurs "dithered" pour le mode 640, sans appel à aucune autre procédure ou fonction:

```
procedure CreerDithPatterns;
var pat, i, patternvalue : Integer;
begin
  for Pat:= 0 to 15 do begin
    patternvalue:= pat * 16 + pat;
    for i:= 1 to 32 do
      gDithPatterns[pat][i] := Patternvalue;
  end;
end;
```

La procédure ci-dessus utilise des variables locales *pat, i, patternvalue,* ainsi qu'une variable globale *gDithPatterns[pat][i]* qui est un tableau de tableaux.
• une procédure issue de *SelectSys*, qui fait appel à des procédures ou fonctions précédemment déclarées ou appartenant à la ToolBox.

```
procedure EffacerRectangle(theNumero:integer;
couleur: integer);
var r: rect; oldport: grafportptr;
begin
  oldport:=getport; setport(gwindowptr);
  setDithColor(couleur);
  r:= PosRect[theNumero];
  Paintrect(r);
  setDithColor(0);
  setport(oldport);
end;
```

3.2. Les fonctions

Une fonction est une procédure un peu particulière. Ce que nous venons de dire des procédures s'y applique donc, en particulier la question des directives. Ce qui différencie une fonction d'une procédure, en ce qui concerne son écriture dans le programme, se trouve:
• dans l'en-tête, qui comporte le mot *function* au lieu de *procedure*, et, à la suite des éventuels paramètres, la déclaration du type du résultat fourni par la fonction;

▶

• dans le corps, où l'on doit inclure une instruction qui fixe, par une assignation, la valeur de la fonction après son exécution. Le plus souvent, cette instruction se trouve à la fin du bloc des instructions.

Ici encore, les exemples ne manquent pas dans les sources en Pascal. Nous en extrayons quelques uns:

• Fonction extraite de *AuxSelect1*, permettant de prendre la valeur d'un contrôle dont on connaît le numéro d'identification. Cette fonction fait appel à deux fonctions de la Toolbox:

```
function GetTheValue(theWindow:WindowPtr; the-
controlID:longint):integer;
var theControlH: CtlRecHndl;
begin
  TheControlH:= GetCtlHandleFromID(TheWin-
dow, theControlID);
  GetTheValue:=GetCtlValue(TheControlH);
end;
```

• Fonction extraite de *SelectSys*, qui prend la date et l'heure et les met sous forme d'une chaîne utilisable partout dans le programme:

```
Function DateHeure: str255; (*lit la date et la place
dans la chaine date*)
var a,i: integer;p:ptr;sa:str255;
begin
  p:= @sa[1];
  ReadAsciiTime(p);
  sa[0]:= Chr(18);
  for i:=1 to 18 do begin (*remettre en ASCII 'infé-
rieur'*)
     a:= ord(sa[i]); sa[i]:=chr(a-128);
  end;
  dateheure:= sa;
 end;
```

L'emploi de *dateheure* comme on le ferait pour une autre variable-chaîne permet d'écrire l'heure exacte, prélevée au moment où le programme en a besoin, soit sur l'écran, soit dans un texte imprimé, soit comme paramètre d'information d'un fichier. Une fonction de ce genre doit être classée parmi des outils standard. On s'en sert quand on en a besoin, et on oublie bien vite qu'on l'a écrite.

3.3. Les paramètres

Chaque procédure ou fonction peut admettre des paramètres. Les paramètres qui figurent dans l'en-tête de la déclaration sont les paramètres *formels* (formal). Les paramètres qui figurent derrière l'identificateur d'une procédure ou fonction lors de son appel portent le nom de paramètres *réels* (actual). TML Pascal II fait la distinction entre quatre types de paramètres: les *valeurs* (values), paramètres ordinaires; les *variables*, précédées, dans la déclaration, du mot-clé *var;* les *statiques*, précédés, dans la déclaration, du mot *static;* les *universels*, précédés de la mention *univ*.

Les *variables* sont des paramètres dont la valeur change pendant l'exécution de la procédure ou de la fonction, et doit être utilisée par la suite.

Les paramètres *statiques* sont une nouveauté de TML Pascal II, et améliorent la compilation. On peut les utiliser quand on est sûr que leur valeur ne change pas pendant l'exécution de la procédure.

Les paramètres *universels* peuvent s'employer pour tout type stocké en mémoire sur une longueur donnée. Ce type s'emploie parfois pour désigner un pointeur. Il acceptera alors les pointeurs de tous types.

Pour tous les paramètres, sachez que si le nombre d'octets que leur stockage demande en mémoire est supérieur à 4, ils sont passés par l'intermédiaire de leur adresse. C'est le cas de nombreux *records* utilisés par la Toolbox, qui sont désignés par leur pointeur. De même pour les chaînes, un paramètre de type *str255* sera représenté, à la compilation, par le pointeur de la chaîne.

3.4. Comment sortir

• La fonction *Exit(nom de procédure ou fonction)* fait quitter la procédure ou la fonction en cours, en se branchant directement à sa sortie, c'est-à-dire au *end;* qui la termine. Voici un exemple extrait de la procédure *SetUpWindow* de *AuxSelect1:*

```
  if _Toolerr <>0 then
     begin
     gErrorNum := _ToolErr;
     DoErrorWind;
     Exit(SetUpWindow);
     end
  else
   MyWindow:=theNewwindow;
end; (*de SetUpWindow*)
```

Il ne faut pas oublier, avant d'appeler *exit*, d'écrire les instructions nécessaires avant de s'en aller. Par exemple, si on a ouvert un fichier, il ne faut pas oublier de le refermer.

• L'instruction standard *Halt* met fin au programme immédiatement, comme si on rencontrait la dernière instruction *End*, qui signifie la fin du programme principal. A employer avec précaution: il s'agit plutôt d'une instruction destinée au programmeur pour lui permettre d'interrompre à sa guise un programme.

Nous ne nous sommes pas préoccupés de la façon dont le compilateur de notre Pascal se débrouillait pour assurer une fin convenable au programme. Il y a des tas de précautions à prendre: remettre les choses en l'état, et appeler la fonction *Quit* de GS/OS. Le Pascal fait cela pour nous, sans trop nous dire comment, et c'est un peu dommage, car avec le GS, la fin d'un programme est forcément le commencement d'un autre.

Nous avons fait le tour des principales instructions à connaître pour écrire un programme en Pascal. Il nous reste encore quelques questions importantes, dont celle des entrées et sorties, qui feront l'objet du prochain article. ■

Revue soft : **GS Numerics**

Une super calculatrice dans votre GS

Stéphane Hadinger

GS Numerics est le programme qui manquait à tous les étudiants et enseignants des sections scientifiques. Il comble le fossé qui existe entre les ordinateurs personnels et les calculatrices scientifiques les plus puissantes, en combinant la puissance de calcul des premiers avec les fonctionnalités étendues de ces dernières.

Une abondance de commandes

Lorsqu'on lance GS Numerics, on se trouve confronté à l'impressionnant écran principal et sa multitude de boutons donnant accès aux commandes du calculateur. La partie supérieure contient les fonctions générales que l'on retrouve sur toute calculatrice un peu perfectionnée. L'affichage des résultats est constitué d'une pile opérationnelle de quatre registres (x-y-z-w) et d'un registre Last-x. Les calculs se font en effet en notation polonaise inversée, chère aux calculatrices fabriquées par Hewlett-Packard. Pour faire 2+3 il faut taper "2", "Enter", "3"et cliquer sur "Add(x+y)". Ce système est un peu déroutant quand on n'a pas l'habitude, mais il s'avère à la longue très pratique et très fiable lorsque les calculs deviennent complexes. Toutefois pour ceux qui sont réfractaires à ce système, il leur est toujours possible de conserver le système classique avec des parenthèses et de taper les formules 'comme dans les livres': "2+3"et "Enter".

Il est à noter que la précision des calculs de GS Numerics est remarquable. Cette application utilise l'outil SANE de la toolbox et profite donc de calculs en précision étendue sur 80 bits. A l'utilisation cela se traduit par une précision de 18 décimales qui surpasse largement toutes les calculatrices scientifiques du marché. Les autres commandes sont plus classiques: conversion d'angles, sélection du format d'affichage en mode fixe ou scientifique, une mémoire de 26 registres.

J'ai cependant une critique à faire en ce qui concerne l'affichage des résultats des calculs, le registre x est affiché avec la même taille que les autres registres et se cache dans un coin de l'écran. J'aurais aimé un affichage plus lisible, plus gros et mieux situé.

Dans la partie centrale de l'écran principal se situent les fonctions scientifiques: log, exponentielle, calcul sur les heures, trigo, trigo hyperbolique et fonctions inverses. Pour couronner le tout, GS Numerics est capable de calculer sur des nombres complexes, qu'ils soient rentrés sous forme cartésienne ou polaire. La pile opérationnelle est alors réduite à deux nombres complexes, les parties réelles sont dans x et z, les parties imaginaires dans y et w. Il suffit alors de double cliquer sur les boutons des fonctions pour calculer en complexe.

Enfin les physiciens trouveront leur bonheur dans la partie basse de l'écran. Ils disposent de boutons de conversion d'unités (37 au total) recouvrant les domaines de longueur, aire, volume, masse, puissance et température. En cliquant sur *Eléments*, une table des 117 éléments chimiques vient remplacer les conversions d'unités. Cette table rassemble tous les éléments dans l'ordre alphabétique de leur nom anglais, de l'actinium au zirconium. En cliquant dans un élément on place sa masse atomique en u.m.a. dans x, et en double cliquant on place son numéro atomique.

GS Numerics offre toutes les fonctionnalités d'une calculatrice scientifique de haut niveau, mais il ne s'agit là que de la partie émergée de l'iceberg. Cette excellente application offre en plus tout un éventail de commandes d'analyse numérique et de résolution de systèmes d'équations.

Polynômes et fonctions

GS Numerics offre des commandes puissantes d'analyse numérique sur les polynômes et les fonctions. Parlons d'abord des polynômes, ils sont uniquement à coefficients réels mais peuvent être tapés soit coefficient par coefficient, soit racine par racine. On peut les évaluer pour un x donné, les dériver ou les intégrer, et, c'est la partie la plus intéressante, calculer toutes leurs racines réelles ou complexes.

En ce qui concerne les fonctions, celles-ci



sont obligatoirement réelles d'une variable réelle. On peut calculer la pente de sa courbe représentative en un point mais il ne s'agit que d'une évaluation numérique pas toujours très précise. On regrettera l'absence d'une dérivation symbolique qui aurait donné de meilleurs résultats, ce que font certaines calculatrices très haut de gamme, mais c'est très difficile à programmer.

Une commande permet la recherche d'une racine d'une fonction avec une précision variant entre 10e-6 et 10e-18, selon le temps que l'on est prêt à sacrifier. Cependant la méthode de recherche de racine utilisée est étonnamment rudimentaire pour un tel programme, elle impose en effet d'encadrer la racine recherchée par deux valeurs dont les images par la fonction sont de signes contraires. Il s'agit là d'une contrainte assez forte qui impose pratiquement de tracer la courbe au préalable et d'encadrer manuellement les racines recherchées. En outre les racines d'ordre pair ne peuvent pas être trouvées. L'algorithme employé semble être une simple dichotomie, méthode qui donne à coup sûr des résultats mais qui ne brille pas par son originalité et sa commodité d'emploi.

Une commande d'intégration numérique est également proposée. Elle utilise au choix la méthode de Romberg, Simpson ou des trapèzes.

La section analyse fonctionnelle offre tout ce dont on a généralement besoin et devrait suffire dans 90% des cas, mais les méthodes utilisées sont très classiques, faciles à programmer et manquent totalement d'originalité. J'aurais aimé avoir le choix entre différentes méthodes de recherche de racines, des fonctions complexes d'une variable réelle (voire complexe), une intégration de Gauss et une évaluation d'intégrales impropres. Vous me trouverez peut-être très difficile mais les auteurs avaient là une occasion de nous montrer leur talent, ils se sont juste contentés de faire le minimum. Je qualifierai cette section de suffisante mais horriblement banale.

Matrices et systèmes linéaires

J'aurais peu de choses à dire sur les commandes de calcul matriciel. Les matrices, d'une taille maximale de 10x10, sont soit réelles soit complexes, et on peut les additionner, les multiplier, calculer leur déterminant et leurs inverses.

Comme extension du calcul matriciel on trouve la résolution de systèmes d'équations linéaires réelles ou complexes, avec une saisie simplifiée dans le cas de matrices symétriques.

Cette section est puissante, complète et bien faite.

Régressions

Lorsque l'on manipule une série de données sous forme de points (x,y), il est souvent intéressant d'essayer d'approcher ces points par une fonction classique en vue d'une interpolation ou d'une extrapolation. Les commandes de régression permettent d'approcher ces points par l'une des quatre fonctions suivantes: y=b.x+a, y=b.ln(x)+a, y=a.exp(b.x), y=a.x^b selon la méthode des moindres carrés. Le choix de la fonction peut-être manuel ou automatique, et on peut tracer la courbe représentative de la fonction obtenue pour la comparer au nuage de points.

Section graphique

Vous l'avez sûrement deviné, GS Numerics est évidemment capable de tracer des courbes mais uniquement sous forme cartésienne, il ne gère ni les courbes en polaire, ni celles en paramétrique. La sélection des échelles est automatique. De plus on peut accéder à toutes les commandes des sections vues précédemment directement à partir du module graphique: calculer la pente de la courbe en un point, intégrer la fonction entre deux points de la courbe, chercher une racine de la fonction en l'encadrant par deux points situés de part et d'autre de l'axe des abscisses... C'est donc le complément idéal des autres sections d'analyse fonctionnelle.

Conclusion

GS Numerics est une application très complète qui ravira les universitaires et les étudiants des classes préparatoires scientifiques, ainsi que leurs enseignants. Il offre toutes les fonctions indispensables avec une grande précision de calcul qui se paye hélas en temps de calcul. La Transwarp GS est conseillée.

Je souhaiterais pourtant voir ajouter dans les futures versions un usage moins lourd des nombres complexes, une extension (dans la mesure du possible) des fonctions et des polynômes au domaine complexe, et pourquoi pas une dérivation symbolique. Le GS pourrait alors concurrencer dans ce domaine les calculatrices haut de gamme de Hewlett-Packard.

GS Numerics est un bon produit, complet et bien présenté. Sa documentation est d'une clarté exemplaire et contient de nombreuses copies d'écran. Il ne s'adresse évidemment qu'à un nombre restreint d'entre vous, mais je le recommande, car c'est un bon outil de travail et surtout un excellent outil pédagogique. ■

à propos de Resource Converter :

Codage et décodage de ressources

Bernard Fournier

Il y a quelques temps de cela, mon attention fut attirée par les pages 21-26 du chapitre Resource Manager du volume 3 du *ToolBox Reference*. En quelques lignes il est expliqué que le Resource Manager permet d'inclure des routines de codage/recodage pour les ressources d'une application. Ainsi, lorsqu'une ressource est spécifiée *resConverter* (bit 11 des attributs: voir ToolBox Mag 3 page 39), alors tout appel à cette ressource (comme *LoadResource, Write Resource*....) est détourné vers la routine de codage/recodage. En clair, cela signifie que le contenu de la ressource sera chargé-décodé ou codé-sauvé à chaque manipulation. On peut donc imaginer une application comme *PicMasterGS* (voir ToolBox Mag 2) ayant une routine de compactage/décompactage d'images, celles-ci étant stockées compactées sous forme de ressources et le codage/décodage se faisant automatiquement lors des appels aux ressources.
On peut même aller plus loin et imaginer qu'une partie du code exécutable de l'application sera stockée sous forme de ressource; et le Resource Manager fera tout seul le travail de chargement/relogement. Cela existe depuis toujours sur Macintosh, et cela est prévu pour le GS. En effet, la ressource *rCodeResource* (type $8017) et la possibilité d'emploi de *resConverter* ont pour but de permettre le stockage du code exécutable sous forme de ressource.
Dans cet article, je n'aborderai que la partie codage/décodage de données; ce qui concerne le codage/décodage de code exécutable étant pour le moment un mystère pour moi (qui peut m'indiquer où se procurer une documentation sur le relogement via *InitialLoad* ?). De toute façon, le principe est le même dans les deux cas, ce qui change c'est le contenu de la routine de conversion. Et lorsque Complete Pascal 2.0 sera disponible, la routine de chargement de *rCode Resource* sera fournie avec le logiciel, ce qui évitera d'avoir à écrire la nôtre !

Comment ça marche

Lorsqu'on charge ou sauve une ressource, le Resource Manager vérifie s'il doit employer une routine de conversion en scrutant le paramètre *resConverter*. La routine de conversion est affectée à un type de ressources grâce à l'appel *ResourceConverter*. Il est à noter que la même routine de conversion peut être affectée à plusieurs types de ressources en employant autant d'appels à *ResourceConverter*.

Le Resource Manager accepte deux sortes de routines de conversions: celles qui seront disponibles pour l'application en cours (liste maximale de 10922 routines différentes) et les routines système qui seront accessibles par toutes les applications. Lors d'un appel à une routine de conversion, le Resource Manager va d'abors scruter la liste des routines de l'application puis celles du système.
Puis il va empiler un certain nombre de paramètres que votre routine devra utiliser pour coder/décoder la ressource. En fin de routine, l'accumulateur devra être positionné à 0 si tout s'est bien déroulé ou à toute autre valeur pour signaler une erreur.

Une routine de conversion peut posséder deux fonctions: *ReadResource* et *WriteResource* (dans ce cas on emploie en plus *ReturnDiskSize*). La fonction *ReadResource* sera appelée lors des chargements de ressource: elle devra donc décoder les informations et les stocker en mémoire. La fonction *WriteResource* ira d'abord vérifier la taille sur disque (par *ReturnDiskSize*) puis procèdera à la sauvegarde de la ressource: la fonction devra donc coder les informations et les stocker sur disque.

ReadResource: une ressource faisant appel à une routine de conversion sera chargée comme suit:
- on spécifie la routine employée avec *ResourceConverter*
- on charge la ressource avec *LoadResource*
le Resource Manager va alors directement passer dans la routine de conversion pour procéder au chargement de la ressource. Cette routine devra donc se charger de lire les informations sur disque (appel à *ReadGS*) et de les décoder avant de les stocker en mémoire. Lors de l'appel, le Resource Manager empile les paramètres suivants sur la pile:
espace (long): contiendra le résultat de la fonction
convertCommand (word): commande employée
convertParam (long): pointeur sur le param-bloc de lecture
resPointer (long): pointeur sur le resource record
et bien entendu en fin d'appel, on trouvera sur la pile le résultat du traitement stocké sur quatre octets (0 si tout va bien, ou code d'erreur survenue).

Le *ResPointer* va permettre de lire les informations relatives à la ressource: son implantation mémoire et sa taille (ce qui permettra d'ajuster le handle qui va recevoir la ressource décompactée). On vérifie- ▶

ra également si la ressource doit être chargée à une zone absolue (indicateur *resAbsLoad*) et enfin on mettra à jour le champ *resHandle* du ResRecord si besoin est (par exemple si la ressource chargée a été transférée dans un nouvel handle suite au décodage).

ConvertParam contient le pointeur sur le param-bloc de lecture à passer en paramètre à *ReadGS*.
ConvertCommand indique si on va faire une lecture (valeur 0), une écriture (valeur 2) ou une vérification de taille (valeur 4).

Note: pour *WriteResource*, le processus est le même, excepté qu'on fait appel à *WriteGS* et que *ResCommand* contient la valeur 2. Il faut également savoir que lors de l'appel d'écriture, le Resource Manager fait un appel à *ReturnDiskSize* pour déterminer la place occupée sur disque: il faudra donc traiter systématiquement cette fonction en cas d'écriture via *Resource Converter* (pour *ReturnDiskSize*, on a les mêmes paramètres avec les différences suivantes: *convertCommand* est indéfini, *convertParam* contient la valeur 4 et la fonction retourne la taille en octets utilisée sur disque).

un exemple en ORCA Pascal

Sur la disquette de ce numéro se trouve un dossier /ResConv.ORCA contenant le source d'un exemple de conversion de routine. A titre d'illustration, j'ai choisi de traiter la conversion d'une chaîne pascal en minuscules. Une fois qu'on a compris le mécanisme de traitement de la ressource, on peut fort bien écrire toute sorte de codage/décodage relatif aux ressources de son choix.
Il sera peut-être nécessaire à certains de se reporter à mon article *Pascal et Assembleur* du n°5.
Dans les déclarations, on indique l'emploi de la procédure *ConvertProc* en tant que module externe, et qui plus est concernant une fonction de la toolbox (d'où l'emploi des directives *DataBank*):

```
{$ToolParms+,$DataBank+}
PROCEDURE ConvertProc;          EXTERN;
{$ToolParms+,$DataBank-}
```

Ensuite examinons la procédure *DoNew* de *ResConv.pas*:
- on commence par spécifier le type de ressources à convertir (dans ce cas les ressources de type $1006):

```
ResourceConverter(@ConvertProc, $1006, resLogIn+
resLogApp);
```

- on signale que la ressource n°1 utilisera la conversion:

```
SetResourceAttr(resConverter,$1006,$0001);
```

- puis on charge la ressource

```
gTxtHandle:=LoadResource($1006,$0001);
```

- comme on traite une ressource de type chaîne pascal, on va déréférencer le handle vers un pointeur de chaîne:

```
gTxtPtr:=pStringPtr(gTxtHandle^);
```

- la suite consiste à afficher éventuellement un message d'erreur ou la chaîne convertie.
Lors de l'appel à *LoadResource*, le Resource Manager va automatiquement faire appel à la routine de conversion *ConvertProc* spécifiée dans l'appel *ResourceConvert*. Cette routine est détaillée dans le source *ProcConv.Asm*. Cette routine est construite comme suit (et servira de modèle pour d'autres routines de conversion):
- on réserve de la place sur la pile et on transforme la pile en page directe:

```
tsc
sec
sbc  #$0004
tcs
phd
inc  a
tcd
```

- ensuite on va mettre à 0 la zone de retour de la fonction (cette zone ne sera changée ensuite qu'en cas d'erreur): on poke $0000 en $11 de la page directe:

```
stz $11
stz $13
```

- on teste quelle est la fonction nécessaire: le paramètre *ConvertParam* contient le code 0 pour *Read*, 2 pour *Write* et 4 pour *ReturnDiskSize*:

```
lda $10
beq Readcv
```

(ici on ne teste que *Read*, mais on peut fort bien ajouter les tests correspondant à *ReturnDiskSize* et *Write* si on implémente ces routines de conversion en écriture).
- une fois analysé le code de la fonction à employer, on se branche sur la routine correspondante. Dans notre cas on va lire la ressource. Tout d'abord on récupère le handle du ResRecord et on récupère le handle de la ressource (10° octet du ResRecord) ce qui nous permet alors de verrouiller ce handle par précaution.

▶

Complete Pascal 2.0

La société Complete Technology (qui a racheté les droits d'exploitation de TML, rebaptisé Complete Pascal) vient de faire l'annonce de la sortie imminente de la version 2.0 de leur Pascal. Celui-ci inclura, outre un éditeur de Ressources complet et fonctionnel (ce qui était loin d'être le cas pour TML), la possibilité de mettre du code exécutable dans les ressources *rCodeResource*. Ceci résoudra de manière définitive le problème que j'avais exposé dans ToolBox Mag 5 page 16: on pourra désormais faire appel à des routines assembleur à partir de TML sans se noyer dans l'emploi des *In-Line*.
Cette annonce plus les nouveaux produits Byte-Works (outils de synthèse vocale, nouvelles versions de Pascal et C) et les notes techniques Apple sur rVersion et rComment (voir ToolBox Mag 5) laissent augurer un avenir radieux pour notre GS (sans parler de la toute dernière version 3.2 de MPW sur MacIntosh, qui continue d'offrir la possibilité de compilation croisée sur Mac de sources pour GS).

Les pièges du Loader

Une précision à propos de la remarque de Ph. Manet dans ToolBox Mag 6 page 8, suite à mon article dans ToolBox Mag 4. Philippe pointe le fait que j'utilise *GetName* au lieu de *LGetPathName2* pour récupérer le nom de l'application en cours. Certes, sur le fond il a raison: pourquoi faire compliqué, si on peut faire plus simple? Oui, mais.... en lisant les notes techniques, on apprend que Apple déconseille l'emploi de *LGetPathName* pour les fichiers *ExpressLoad*. Comme l'application en question peut éventuellement être passée à la moulinette *ExpressLoad* par un utilisateur, j'ai opté pour l'assurance tout-risque en employant *GetName*.
Si effectivement un NDA peut avoir son préfixe d'origine différent de celui en cours (cas des NDA chargés par un lanceur de NDA), je vois difficilement comment une application pourrait actuellement subir la même chose... ce qui fait qu'il n'y a nul besoin de connaître le préfixe de l'application en cours (c'est forcément celui en cours puisqu'on travaille lors du chargement) et qu'ainsi seul le nom est important pour pouvoir ouvrir le fichier ressource associé. Il résulte de ceci que les NDA employant des ressources et donc *LGetPathName* ne doivent pas être passés à *ExpressLoad* (et d'une manière générale n'employer cet utilitaire qu'à bon escient). *[NDLR: le même raisonnement vaut pour les utilitaires EXE lancés par un shell, type APW].*

La documentation relative au Loader est disponible sous forme de classeur auprès de APDA ou sous forme de manuel auprès de Addison Wesley: il s'agit de *GS/OS Reference tome 1: Applications and GS/OS* (un tome 2 intitulé *Devices and GS/OS* est disponible uniquement en classeur pour l'instant).

```
phb             permettre l'adressage absolu
phk
plb
ldy   #$0010       GetHandle du Resource Record
lda   [$07],y      on lit le 10ème octet du ResRec)
sta   $00          et on stocke en page directe
iny
iny
lda   [$07],y
sta   $02
                   ; verrouiller le Handle
Pei   $02          on verrouille par précaution
Pei   $00
_HLock
```

- on va maintenant déterminer les paramètres de lecture puis effectuer un ReadGS pour charger la ressource en mémoire:

```
lda   [$00] récupérer le ptr de la ressource
ldy   #$0004
sta   [$0B],y   stocker ce ptr dans le buffer Read
ldy   #$0002
lda   [$00],y
ldy   #$0006
sta   [$0B],y
Pei   $0D
Pei   $0B
Pea   ReadGs
Jsl   >GSOS   on lit la ressource avec ReadGS
```

- ensuite on va décoder la ressource: dans notre cas on lit la chaîne caractère par caractère et on effectue un ORA afin de transformer les caractères en minuscules. Cet exemple très simple permet de montrer comment récupérer le contenu de la ressource, ceci peut donc servir de base pour créer des routines plus élaborées (compactage d'images, de sons, de texte...)
- désormais la ressource est décodée et se trouve en mémoire, on quitte donc notre routine en restaurant la pile et la page directe originale.

Cet exemple, d'une portée limitée, est en fait l'ossature de toute routine de conversion. Une fois que l'on a compris comment récupérer le handle, le pointeur et le contenu de la ressource à décoder, on peut écrire sans problème ses routines personnelles. Il faut juste garder présent à l'esprit que tous les paramètres sont sur la pile qui sert de page directe; après ce n'est qu'une affaire d'adaptation à ses propres besoins.

un exemple en TML Pascal

Sur la disquette de ce numéro se trouve un dossier intitulé /ResConv.TML contenant le source de la conversion de ressources à partir de TML. Comme nous l'avions vu dans le n°5, l'adjonction de routines assembleur dans TML ne peut se faire (pour le moment) que par l'emploi de *InLine*. Attendons la sortie de Complete Pascal 2.0 pour pouvoir enfin manipuler les routines assembleur de la même manière qu'avec Orca. *[NDLR: voir aussi la solution de Stéphane Hadinger dans ce numéro].*
Tout ce qui concerne la déclaration d'utilisation d'une routine de conversion par *ResourceConverter* se fait de manière similaire à ORCA. Cependant, la déclaration de la procédure *ConvertProc* se fait au sein même du source TML en n'omettant pas de spécifier qu'il s'agit d'une procédure s'appliquant à une fonction de la toolbox (emploi des directives *{$DefProc}*).
Dans ToolBox Mag 5, nous avions vu que TML fait appel à ses procédures en réservant immédiatement de la place sur la pile, puis en écrivant le code et à la fin il restaure le pointeur de pile (voir l'article en question pour de plus amples informations). Dans notre cas, il va donc falloir restaurer la pile immédiatement avant notre code de conversion et quitter notre routine par un RTL court-circuitant la fin de procédure générée automatiquement par TML.
En ce qui concerne l'annulation de la réservation de place sur la pile, il suffit de restaurer les registres dans l'ordre inverse de leur modification par TML. Si on se réfère à la page 17 de ToolBox-Mag 5, il faut donc écrire le code suivant:

```
{annuler la séquence d'initialisation de TML}
```



```
plb
tdc
clc
adc     #00FA
tcs
pld
```

Ensuite on écrit notre code de chargement/conversion (le même que pour ORCA) en utilisant les directives *InLine* du module *AsmAndPas.p.o* (contenu dans le dossier /ResConv.TML) et en respectant les consignes énoncées dans ToolBox-Mag 5. Je rappelle qu'on quitte la routine par RTL puisqu'on a annulé le code d'accès de la routine généré par TML (on n'a donc pas besoin du code de sortie).

compilation des sources

Les utilisateurs de TML devront ajouter l'unit *AsmAndPas.p.o* dans le dossier des librairies et signaler dans les *Uses* l'emploi de ce module. Ensuite il suffit d'étudier le listing du source principal (je signale que ce source est à quelques lignes près celui généré par Genesys, mais il a certains bugs corrigés et certaines procédures indispensables au fonctionnement ont été ajoutées).
Indiquer au compilateur qu'on emploie les ressources de *ResConv.rez* et compiler puis exécuter l'application.
note: on peut aussi modifier les ressources avec un éditeur (Genesys par exemple) ou avec REZ de APW en recompilant les sources (*compile ResConv.src keep=$.rez*).

Pour les utilisateurs d'ORCA, il y a deux méthodes:
- pour ceux qui disposent d'un assembleur, lancer le script *MakeConv*.
- sinon, modifier le script *MakeConv* de la manière suivante:

```
show time
delete ResConv
compile +L +S +E ResConv.pas keep=$
link +l +W ResConv ProcConv 2/paslib 2/syslib keep=$
duplicate -r ResConv.rez ResConv
ResConv
```

On pourra également modifer les ressources de *ResConv.rez* avec un éditeur ou directement sur les sources si on compile avec APW/REZ (lancer le script *ResConvRez*).

Si vous compilez les sources des ressources, il faut bien faire attention à indiquer le préfixe d'accès à *Types.rez* et *Types.rez.aux* (cette librairie est située dans le dossier */Resconv* et doit être placée avec *Types.rez* dans le dossier *R.Libraries* d'APW).

contenu du dossier /ResConv

- *Note.ResMap*: un fichier texte à lire
- la dernière version de la librairie *Types.rez.aux*
- le dossier */ResConv.Orca*, qui contient:
- *MakeConv* script de compilation du programme
- *MakeConvRez* script de compilation des ressources
- *ResConv.rez* les ressources
- *ResConv.src* source des ressources
- *Window.src* source des ressources
- *Resconv.pas* source principal en Pascal
- *ProcConv.asm* source de la conversion en assembleur
- *ProcConv* routine compilée prête à être linkée
- *ResConv* application à exécuter
- *LinkAll* et *Macros* servent à la compilation de *ProcConv.asm*
- les fichiers *.a* et *.root* sont générés par APW/ORCA
- le dossier */ResConv.Tml*, qui contient:
- *AsmAndPasU.p* source des *InLine* assembleur
- *AsmAndPas.p.o* unit compilée des *InLine*
- *ResConv.rez* les ressources
- *ResConv.src* source des ressources
- *Window.src* source des ressources
- *ResConv.p* source TML de l'application
- *ResConv* application à exécuter ■

Pas de panique!

En mettant au point la routine de conversion illustrant cet article, j'avais sans cesse une erreur $308 lors de l'exécution de mon application. Je pensais donc que ma routine de conversion était incorrecte, et je planchais des heures durant sur la documentation à la recherche du détail qui m'aurait échappé. Peine perdue, cette @#!¿º d'erreur $308 se produisait systématiquement. Sur le moment, je fus tenté de crier au scandale en incriminant TML de tous les maux. Mais avant de protester, je recopiai une version inutilisée de TML sur mon disque dur et procédai à une compilation: et cette fois cela marchait! Le compilateur avait donc été modifié à un moment ou un autre, et tous mes ennuis provenaient de là.

Lorsqu'une application présente des comportements inexpliqués, il faut procéder comme suit:
• en cas de développement, vérifier que le source ne contient pas d'erreur de programmation.
• ôter les NDA et inits du système et vérifier si les perturbations persistent.
• recopier une version jamais utilisée du programme incriminé sur une disquette ou disque dur et refaire un test (ceci montre l'importance de conserver en sûreté ses logiciels originaux, et par conséquent de pouvoir en faire des copies sans problème. La seule attitude face à un logiciel protégé anti-copie: *ne pas l'acheter*).
• ré-installer le système GS/OS sur le disque dur ou sur la disquette de boot: cela suffit parfois à tout remettre dans l'ordre…

"Incompatibles" :

NeXT ... is now !

Jean-Michel Vallat

J'ai été très surpris lorsque Jean-Yves m'a demandé un article sur NeXT pour ToolBox-Mag. Ancien possesseur de IIe et de IIgs, j'avais cru quitter un univers woznakien peuplé de quelques courageux fanatiques irréductibles et inconvertibles… Voici donc ce que je peux dire sur le présent et le futur de NeXT après quelques mois d'utilisation d'un NeXTcube. Pas de polémique NeXT/Apple, pas de polémiques non plus sur Steve Jobs ou sur l'aspect "réussite commerciale" de NeXT: des commentaires, des découvertes, et… de la passion.

Compatibilité ou progrès?

L'expérience…

Depuis le sommet des hautes falaises des bords de mer, on peut voir le monde moderne nous offrant tout le confort moderne: hôtels, restaurants, piscines… C'est le High-Tech, le dernier cri; c'est ce que l'on voit dans les publicités, ce dans quoi les businessmen sont prêts à investir. C'est beau, un peu cher, mais surtout c'est éphémère: après quelques années d'existence, tous ces produits sont balayés par le vent du progrès, irrémédiablement.
Vue de la plage, notre falaise offre un spectacle tout autre. On peut y voir les différentes couches formées au cours de l'histoire par la nature et le temps: de la préhistoire jusqu'au XXème siècle, tout y est visible.
Eh bien, il en est ainsi dans le monde des compatibles PC. Dans les revues et dans les magasins, on peut facilement acheter toutes les dernières productions du Far-East pour des prix, somme toute, très raisonnables. Mais chez les particuliers, dans les entreprises et les écoles, il n'est pas rare de découvrir des monstres préhistoriques: 8088 à 4 MHz se dressant fièrement aux côtés de lecteurs 360 Ko, et d'écrans CGA. Un monde à deux vitesses donc, qui ne plait à personne, mais qui existe.
Ce qu'on voit là, ce sont **la force et la faiblesse d'une base installée.** La force, c'est le nombre, un marché titanesque qui permet aux fabricants d'investir des sommes gigantesques avec un maximum de chances de retomber sur leurs pieds. La faiblesse, c'est l'extraordinaire inertie de ce monde PC. Tout ce qui existe aujourd'hui porte à coup sûr la marque du passé. Les logiciels et les cartes qui sont commercialisés aujourd'hui contiennent une multitude d'éléments totalement inutiles pour leurs acquéreurs, mais nécessaires pour la compatibilité. Il n'est pas rare de voir des dizaines de "drivers" d'écrans four- ▶

Incompatibles

Apple n'a pas tort, dans le fond, de rappeler en permanence aux utilisateurs de GS qu'il n'y a pas que l'Apple II. Effectivement, nous vivons dans un monde où aucun ordinateur ne peut plus prétendre être le seul, et les utilisateurs de GS ont, comme les autres, envie de savoir ce qui se passe dans "le monde extérieur". Vus du GS, les micros du monde extérieur n'ont qu'un point commun: ils sont incompatibles GS. "**Incompatibles**", tel sera donc le titre de cette nouvelle rubrique. Les incompatibles d'Apple lui-même, appelés Macintosh, y viendront, à leur place — qui est bien minime: dans ce continent, Apple n'est qu'un **tout petit** canton.
Il faut bien commencer par quelque part: nous avons choisi non pas le plus courant des incompatibles, mais tout simplement **le meilleur**, celui qui a été conçu par l'homme qui avait, entre autres, déjà conçu l'Apple IIc: Steve Jobs.
Qui pouvait mieux nous en parler que Jean-Michel Vallat, un des auteurs du magnifique jeu **Bouncing Bluster** sur Apple IIGS? "Incompatible", bien sûr, le NeXT a pourtant, Jean-Michel en fait la preuve, quelque chose en commun avec le GS: c'est une machine qu'on **aime**…

JYB

pas rare de voir des dizaines de "drivers" d'écrans fournis pour un logiciel donné: drivers CGA, EGA, PGA, VGA, AGAGA, etc. Un PC, même le plus récent, est la plateforme idéale pour effectuer une thèse sur les technologies des dix dernières années…
Le problème est encore plus grave lorsqu'une machine reste trop longtemps construite autour d'un même processeur: après 3 ou 4 ans d'existence, les machines voient leur catalogue de logiciels s'emplir d'une foule de programmes en tout genre, plus originaux les uns que les autres (cf. l'Apple II par exemple). Et, suivant les recommandations et le dynamisme des contructeurs de ces machines, la situation tendra à se calmer ou au contraire, empirera de sorte que bientôt il sera impossible de changer. C'est le cas des ST et autres Amigas qui se voient aujourd'hui quasiment incapables d'évoluer d'une façon significative, sous peine de perdre 80 à 90% des logiciels disponibles.
Du côté du Mac, grâce à une politique très stricte et à la sortie régulière de nouvelles versions de son système, Apple a su plus ou moins calmer le jeu. Les développeurs ont compris qu'il était dans leur intérêt de "respecter les règles du jeu" sous peine de devoir recommencer leur produit régulièrement, et au risque de perdre tout aussi régulièrement leur crédit auprès de leur clientèle. Reste que, dans le Macintosh originel, tout était propre et beau, et… monochrome. Le Mac n'avait pas été pensé au départ comme une machine 24 bits couleur. Conséquence: avec la couleur et QuickDraw 32, incompatibilités et bombes en tout genre sont le lot quotidien des utilisateurs de Mac.

… et ses leçons

NeXT, en arrivant bon dernier, profite de toutes les dernières innovations matérielles et logicielles, et… des leçons de l'expérience. Ses machines ont été conçues pour pouvoir évoluer au moindre coût, et sans perte de compatibilité. Par exemple, tout ce qui concerne l'affichage écran est sous le contrôle de Display PostScript: quelles que soient les caractéristiques de votre écran, le nombre de couleurs, vous pourrez toujours utiliser l'ensemble des logiciels tournant sous NextStep, sans souci de problèmes de résolution d'écran. Libération pour les développeurs, tranquillité pour les utilisateurs.
NeXT en est déjà à sa seconde génération de machines (processeur 68040 au lieu de 68030) et de système (NextStep 2.0). Lorsque l'on regarde tout ce qui a été changé, tant au niveau hard qu'au niveau soft, entre les deux générations, on est assez étonné de voir des logiciels réalisés en 1989 sous NextStep 1.0 fonctionner sans problème aujourd'hui en 2.0. Pourtant, il serait faux de prétendre que la version 1.0 (1988) et la version 2.0 (1990) sont complètement compatibles. Mais les évolutions vont à chaque fois dans le bon sens: la simplification et l'amélioration de NextStep. En fait, à quelques détails près, tous les logiciels du 1.0 tournent en 2.0 (le contraire n'étant, bien entendu, pas vrai).
C'est que NeXT a été concu dès le départ sur un principe très simple, qui concilie compatibilité et progrès technique: réaliser un système indépendant du matériel (processeurs, etc) et du logiciel (système d'exploitation). C'est pour cette raison que l'on trouve NextStep au catalogue d'IBM pour ses stations RS/6000. Et c'est aussi pour cela que la prochaine génération de NeXT sera (probablement) basée sur les processeurs de la famille 88000 de Motorola. Quant aux applications, elles sont simplement programmées pour tourner sous NextStep, sans savoir sur quelle machine elles tournent effectivement.

Le graphisme et les couleurs

Display PostScript, la compatibilité assurée

Sur le GS, la valeur d'un pixel de l'écran correspond à une valeur (codée sur 2 ou 4 bits: 4 ou 16 couleurs) dans une palette de couleur, qui contient elle-même une valeur (codée cette fois sur 12 bits: 4096 couleurs). Cela donne 16 couleurs parmi 4096 à l'écran.
Display PostScript Couleur de NeXT ne travaille pas avec une palette de couleurs: à chaque pixel est associée une valeur correspondant à une couleur. Lorsque vous ouvrez une image "4 bits" sur une NeXTdimension (qui travaille à l'affichage en 32 bits), Display PostScript réalise "automagiquement" la conversion et simule les couleurs de l'image. Et le plus beau, c'est que ce principe est aussi valable dans l'autre sens: pas de problème pour ouvrir une image 32 bits sur une NeXTstation (4 bits monochrome). L'image obtenue sera la simulation la plus fidèle qu'il est possible de réaliser avec seulement 4 gris, et très honnêtement, il faut voir le résultat pour bien comprendre tout le génie de cette technique.
En ce qui concerne la résolution de l'écran, là encore, Display PostScript permet de répondre définitivement: PostScript est un langage de description de page vectoriel. PostScript imprime déjà le même fichier de la meilleure façon possible aussi bien sur une LaserWriter à 300 points par pouce que sur une photocomposeuse à 3500 points par pouce. De la même façon, si dans un avenir plus ou moins proche NeXT propose des écrans à 200 points par pouce (5 fois plus de points!), les logiciels actuels continueront de fonctionner avec des surfaces de travail identiques (à l'écran) mais avec des tracés beaucoup plus fins…
[Restera a résoudre le problème des images TIFF (bitmap, pas vectorielles) que l'on retrouve au niveau de l'interface graphique (icônes): ces images sont dépendantes de la résolution de l'écran, et par conséquent se verraient réduire d'une façon alarmante en cas d'augmentation du nombre de pixels. Gageons simplement que les futures machines seront suffisamment puissantes (la génération actuelle en est déjà parfaitement capable) pour calculer et déformer en temps réel ces images et leur redonner leur taille d'origine.]

Quatre couleurs… ou des milliers?

Dans leurs versions de base, les NeXTstations et NeXTcubes sont monochromes (en fait 4 nuances de gris). Pourtant, lorsque l'on regarde un écran NextStep, il est difficile de croire que ce que l'on voit est le fruit d'un système disposant de si peu de couleurs. La raison ? Eh bien, comme sur le GS en mode 640, les NeXT utilisent l'effet de "mélange" obtenu par la proximité des points à l'écran (le "**dithering**"). Les écrans MegaPixel des NeXT possédant tous une résolution de 1120 par 832 avec une densité de 92 points par pouce, cet effet est réalisé non pas grâce à un effet de flou, mais grâce à la finesse des points. NeXT a poussé la perfection jusqu'à mettre au point une technique permettant de simuler la couleur. Dénommée "Compositing" et intégrée de façon transparente à Display PostScript, cette technique permet aussi de réaliser des effets logiques (13 au total) entre les couleurs lors de ▶

l'affichage d'une région: addition, soustraction, xor, source devant, destination derrière, et ainsi de suite.
On appelle **transparence graphique** la technique consistant à coder, en plus de la couleur de chaque point, une valeur (alpha) représentant l'opacité de cette couleur. Sur les machines NeXT à affichage 4 bits, la transparence est gérée sur 4 niveaux. Là encore, la technique du mixage des nuances de gris permet de simuler un nombre important de plans pour la transparence. Sur la NeXTstation Color, les couleurs sont codées sur 12 bits (4096 nuances) et les transparences sur 4 bits (16 niveaux). Enfin, un cube équipé de la carte NeXTdimension, gère les couleurs sur 24 bits (16 millions de nuances) et les transparences sur 8 bits (256 niveaux).
A la différence du Mac (où l'on retrouve sur certains modèles ce genre d'effets), les NeXT disposent tous de cette notion de transparence, car c'est une fois de plus Display PostScript qui se charge de ces opérations. Dès lors, tous les logiciels sont en mesure d'exploiter ces "Alpha Channels", et surtout ils restent tous compatibles entre eux sur ce point, ce qui n'est pas le cas sur les machines d'Apple.

Musique: la qualité CD

Synthèse ou calcul?

Lorsque j'ai commencé à programmer sur GS, aucun outil sonore sérieux n'était disponible. Tout le monde était prêt à reconnaître les possibilités extraordinaires de l'Ensoniq (le co-processeur sonore), mais personne ne semblait savoir comment s'y prendre pour en tirer quelque chose. Il y avait bien les "Tools" officiels d'Apple, mais ceux-ci restaient le plus souvent mal documentés ou "En Travaux". Lorsque SoundSmith est sorti, j'ai cru entendre "la meilleure musique qu'un micro-ordinateur puisse jouer", comme l'affirmait Joe Kohn dans A+. Le son était peut-être un peu métallique... mais il y avait 14 voix mono, ou 7 en stéréo, et des méga-octets d'instruments disponibles: une sacrée différence par rapport à un bon vieux IIe, et un sérieux concurrent pour le Mac ou l'Amiga. Pourtant, (je pense que vous voyez où je veux en venir) je suis prêt à admettre que je m'étais trompé, et je pense que Joe en ferait de même s'il avait le temps d'écouter un NeXT.
Le responsable de la qualité sonore des NeXT se nomme **DSP 56001** (Digital Signal Processor ou Processeur de Traitement du Signal) de Motorola. En fait, son rôle n'est pas exclusivement musical. Il est utilisé pour effectuer rapidement n'importe quel traitement nécessitant des calculs: vectoriels, matriciels, traitements de l'image ou du son... ainsi de suite.
C'est ainsi que du haut de ses 13 Mips, ce brave DSP est en mesure de reproduire du son de qualité: par exemple, avec 88200 mots de 16 bits traités par seconde, les NeXT sont en mesure de fournir **une qualité sonore identique aux Compact Disks**. Sympathique tout ça !
Mais, me direz-vous, Soundsmith permet de faire de la **synthèse** sonore, c'est-à-dire jouer un morceau de musique à partir d'une partition et d'une simple onde sonore pour chaque instrument. Or, dans mon précédent propos, je mettais en valeur la capacité de restitution d'un morceau de musique **numérisé**: ce n'est pas la même chose. Peut-on donc faire de la synthèse musicale avec un DSP ?
Oui, bien sûr. Pourtant, le NeXT ne dispose pas de "voix" comme le GS. Restituer un morceau numérisé en stéréo ne nécessite que deux canaux. Synthétiser une partition de musique de 10 ou 15 instuments nécessite un grand nombre de voix. Eh bien, c'est là que l'on découvre l'intérêt du choix du DSP par NeXT: plutôt que de prendre un processeur classique pour le son, ces messieurs ont préféré choisir une puce effectuant des calculs rapides, et lui ajouter un convertisseur Digital/Analogique. Ainsi, plutôt que de générer le son de 15 instruments (par exemple) pour deux haut-parleurs, il est bien plus simple de calculer l'onde sonore résultante pour chacun d'entre eux. Ainsi, il n'y a plus de véritable limite au nombre d'instruments aptes à être interprétés simultanément: seuls la complexité et le nombre des calculs qu'il est nécessaire de réaliser peuvent représenter un facteur limitant.
Le programmeur apprendra avec intérêt que NeXT propose un ensemble de routines (méthodes en Objective-C) permettant d'employer les capacités du DSP sans être obligé d'avoir recours à l'assembleur du 56001.

La reconnaissance de la voix.

Certains se souviennent peut-être du retard que NeXT avait accumulé entre 86 et 88, et des critiques de la concurrence à ce propos (en particulier, de la part d'employés Apple). Ce retard avait été causé, entre autres raisons, par des difficultés rencontrées dans la mise au point d'un traitement de textes qui reconnaîtrait la voix humaine. Il s'agissait d'une pièce maîtresse dans la Révolution NeXT, et c'est d'ailleurs pour cela que le premier Cube disposait d'un DSP et d'un micro en standard.
Aujourd'hui **la reconnaissance vocale sur NeXT existe**: c'est l'Université Carnegie-Mellon (en Californie) qui a réussi la première ce tour de force. Dans l'équipe des chercheurs qui ont mis au point ce produit, il y avait un français: Jean-Michel Lunati. Jean-Michel, qui continue ses développements sur ce projet tout en travaillant pour l'un des principaux NeXTcenter de la région parisienne, m'a assuré que bien que le système actuel fonctionne en américain, il n'y aura aucun problème majeur pour en faire une version française. L'ordinateur sans clavier va-t-il enfin devenir réalité? A suivre de près.

Le stockage du son

La qualité CD c'est magnifique, mais cela prend de la place! Si vous n'arrivez pas à remplir votre disque dur de **40** Mo, sachez que c'est la place demandée par un enregistrement de **4 minutes** à 44,1 KHz en stéréo. C'est là où l'on comprend la différence fondamentale entre une cartouche Syquest et un Magnéto-Optique ou un CD-ROM. Heureusement, les prix baissent chaque jour dans le domaine des périphériques de stockage, et il est probable que d'ici quelques mois, le modèle d'entrée de gamme de NeXT ne sera plus proposé avec un disque dur de 105 Mo, mais avec un modèle d'au moins 600 Mo. Les disques de 1 ou 2 Go représenteront des produits normaux, et rares seront ceux qui seront surpris par de tels chiffres.

Le système: multitâches et communication.

Le standard Unix

En 1984, Steve Jobs, avec le Macintosh, avait cherché à mettre au point une machine révolutionnaire et à impo- ▶

ser un nouveau standard. Mais ce qu'Apple et Jobs ont découvert dès les premiers mois après le lancement du Mac, c'est qu'il n'est pas facile d'imposer un produit dont le microprocesseur est plus ou mois inconnu des programmeurs, dont le lecteur de disquettes est totalement incompatible avec les machines existantes sur le marché, et dont les divers connecteurs ne respectent qu'une seule norme: celle du Macintosh.

A l'époque, Apple proposait des solutions originales qui étaient osées et risquées, et que l'on retrouve aujourd'hui sur quasiment toutes les machines (ou presque: voir les premiers paragraphes). Cette attitude avant-gardiste est à la base de la stratégie commerciale et du succès de cette firme. De plus, le temps a résolu une partie des problèmes, et Apple a fait évoluer la machine vers les réalités du marché. Mais cela se passait en 1984, et à cette époque les compatibles IBM ne régnaient pas en maîtres, comme c'est le cas aujourd'hui.

Steve Jobs a donc cette fois-ci écouté les besoins réels des utilisateurs avant de réinventer l'ordinateur personnel des années 90. Or, les points essentiels de la machine NeXT tournaient autour de deux notions: la première NextStep, la seconde une conception différente de l'ordinateur. Pour le reste, il semblerait que NeXT ait préféré jeter un coup d'œil objectif et impartial sur le monde informatique. De cette attitude NeXT a retenu les avantages des PC, des stations de travail, et du Mac. La plate-forme idéale se devait d'être multi-tâches, évolutive, puissante, et simple d'emploi.

Or la référence dans le domaine des systèmes multi-tâche, c'est **Unix**. Ce système d'exploitation très répandu sur les stations de travail, est très peu employé sur micro. De plus, il a la réputation (parfaitement justifiée) d'être tout, sauf simple d'emploi. Le pari de NeXT était donc de réussir à réunir ces deux notions dans un environnement performant. Pari réussi, car il est parfaitement possible d'utiliser la machine sans jamais ouvrir le moindre Shell et taper le moindre ordre Unix (les GS-istes qui ont déjà utilisé APW comprendront l'idée).

L'Unix de NeXT se nomme **Mach** (pure coïncidence!) et a été développé par l'Université Carnegie-Mellon (encore eux). Ses principales qualités se situent dans les domaines de la mémoire virtuelle (qui se trouve être dynamique, et non pas statique comme le font presque tous les autres constructeurs), du multi-processing (capacités à diviser une tâche en plusieurs sous-tâches), et de la communication entre les applications.

La communication entre applications

Pour communiquer des informations entre deux applications, il existe plusieurs méthodes: le presse-papiers (Copier/Coller), ou bien le déplacement d'une icône depuis le Workspace (le super-multifinder du NeXT) directement dans une fenêtre d'une application. Pourtant, cette notion d'échange peut aller plus loin: il est aussi possible de faire faire par d'autres ce que l'on ne sait pas faire, ou, en d'autres termes, **utiliser les programmes des autres pour ses propres besoins**. Ce principe se nomme la **Communication Inter-tâches**. Une application donnée envoie un message (un ordre) vers une autre application, et reçoit une réponse de cette application.

Un exemple: imaginons que, dans un tableur, vous deviez faire un calcul mathématique particulièrement complexe. Il suffira au tableur d'envoyer vos équations vers le logiciel "Mathematica", présent quelque part sur le disque dur ou le réseau, et de laisser faire celui-ci. Idée simple, mais qui risque de simplifier bon nombre de problèmes.

L'Ordinateur Interpersonnel

La révolution d'hier, c'était le Macintosh et son interface graphique intuitive. Cette machine, prévue pour être utilisée par des personnes découragées par la lourdeur et la complexité du monde traditionnel de la micro, n'est malheureusement plus apte à répondre aux besoins de certaines applications, aussi bien qu'à l'emploi qui en est fait par les utilisateurs. Né isolé, le Mac a dû évoluer vers les réseaux et les cartes d'extensions. AppleTalk était une solution intéressante pour transférer des images 1 bit ou de simples textes entre deux Macs, mais il est totalement dépassé pour les besoins actuels: on travaille désormais avec des images 24 bits, des documents de plusieurs dizaines de Mo ou des sons numérisés de taille gigantesque.

C'est pour répondre à ces besoins que NeXT propose en standard sur toute sa gamme de machines des connecteurs Ethernet. Mais c'est aussi pour proposer un nouveau concept: offrir une nouvelle façon de travailler dans les entreprises, à savoir **l'Ordinateur Interpersonnel**. Le Mac permettait un travail performant au niveau individuel; NeXT propose d'offrir le même type de performance, mais au niveau du groupe. De nombreuses applications commencent à apparaître aux USA sur ce thème de l'échange de données et du travail en commun.

Sous le capot

Bien sûr, ni l'utilisateur de NeXT ni le programmeur n'ont à s'occuper de ce que la machine a dans le ventre (microprocesseur, etc). Il faut pourtant en dire quelques mots pour ête complet. Outre le DSP 56001, vous trouverez donc sous le capot d'un NeXT:

• **Un 68040**: dernier-né de la famille des microprocesseurs 68000 de Motorola, ce monstre de concentration est capable de traiter 20 millions d'instructions par seconde (Mips) dans sa version à 25 Mhz; nous sommes bien loin des 5 Mips d'un 68030, et encore plus des 0.2 Mips d'un Apple II !

• **Une architecture DMA** (Direct-Memory-Access): cela consiste à transférer les données entre la mémoire d'une machine et un périphérique sans passer par le microprocesseur de cette machine. Jusqu'a présent, c'était toujours le microprocesseur qui servait d'intermédiaire entre les différentes entrées/sorties et la mémoire, car câbler des chemins d'accès en lecture et écriture entre l'ensemble des différents composants posait de sérieux problèmes de coût et de synchronisation. Mais le moindre accès à un disque dur ou à une imprimante Laser ralentissait, voire paralysait le travail. En fait, le DMA est au hardware, ce que le multi-tâche est au Soft.

Si elle est déjà implantée sur l'Amiga, le Mac IIfx, ou la carte SCSI/DMA du GS, cette technique a été particulièrement poussée sur NeXT: 9 canaux DMA contrôlés par une puce spécialement développée par NeXT (l'ICP ou Integrated Channel Processor). Réseau Ethernet, vidéo, DSP, disques (SCSI/Magnéto-optique), imprimante Laser... tout y passe, sans que le 68040 n'ait son mot à dire.

• **Les Slots** (pour le NeXTcube uniquement): avoir des connecteurs d'extensions sur sa carte mère, c'est bien, et pourtant, ce n'est pas très malin, sachant que c'est l'un ▶

ment (pour des raisons de compatibilité, bien sûr). NeXT a adopté une philosophie différente: placer quatre slots capables de communiquer entre eux dans le cube (NeXT-bus 32-bits), y brancher une source d'alimentation. Ensuite, il suffit d'insérer dans l'un des slots une carte mère et vous obtenez ainsi une machine performante, ouverte, et capable d'évoluer au moindre coût: la mise à niveau des cubes 68030 de 1988 en cube 68040 coûte 8265 F HT.
L'architecture NeXTbus est capable de gérer jusqu'à **15 slots**, chacun pouvant contenir jusqu'à **4096 Mo** d'espace physique. On comprend dès lors que la limitation des modèles actuels à 32 et 64 Mo (respectivement pour les NeXTstation et les NeXTcube) vient du fait qu'il n'existe véritablement sur le marché que des SIMMs de 4 Mo (4 Mo x 8 emplacements et 4 Mo x 16 emplacements). Dès que les prix des mémoires 16 Mo se situeront dans des zones plus abordables, il sera facile de pousser ces machines à 128 et 256 Mo.

L'avenir sourit aux audacieux

L'aventure de NeXT ne fait que commencer, et pourtant déjà l'histoire se répète: en 84, Jobs innovait et se faisait copier par ses principaux concurrents; en 88, Jobs innove et se fait copier par ses principaux concurrents. Voyez les derniers modèles Apple: la touche "Reset" reconvertie en touche "Power" sur certains modèles, les micros greffés sur les LC et SI, l'adoption du standard Ethernet et SCSI-2 pour les futurs modèles, etc.
Pour ceux que l'accord IBM-Apple de Juillet dernier inquiéterait, rappelons que l'histoire d'Apple est déjà jonchée d'un bon nombre d'accords, de prime abord fracassants, mais qui se sont toujours révélés sans véritable lendemain (Texas Instruments et DEC en particulier, en 1988). De plus, les lecteurs de ToolBox-Mag le savent, les utilisateurs ne vivent pas avec les gros titres commerciaux de la presse spécialisée, mais avec des produits réels, performants, et non bogués. Mêmes commentaires pour les développeurs qui ne voient rien de sérieux se mettre en place sur l'horizon Apple dans ces grandes annonces, si ce n'est de nouvelles documentations de plusieurs kilos tous les 6 mois. Même si tout est possible, il serait étonnant qu'Apple réussisse à gagner dans ce rapprochement des notions facilement intégrables dans la gamme de produits Macintosh.
Quant à l'accord Motorola/IBM datant de la même période, il est tout bénéfice pour NeXT, car il se rapporte aux processeurs que l'on retrouve dans les RS/6000 actuels d'IBM. NextStep étant déjà disponible sur cette plateforme, si la puce en question (Power Risc) voit le jour dans quelques années, il y a fort à parier que NeXT s'en servira pour une prochaine génération de machines. A la différence d'Apple qui devra imposer un nouveau standard probablement incompatible avec les Macs actuels, NeXT offrira une plate-forme disposant déjà d'une vaste logithèque.
Certains prétendront que n'importe quel constructeur est capable de faire ce que NeXT propose (depuis déjà de nombreux mois). Mais ils oublient qu'il a fallu le rêve, l'imagination et le charisme de persuasion de Steve Jobs pour faire sortir des laboratoires et des universités toutes les innovations de NeXT, et tout cela dans le plus grand secret et la plus grande passion: **une passion ne se monnaie pas, elle se crée.**

■

MidiSynth et les Beatles

- Pourquoi le Tool MidiSynth ne sera-t-il pas inclus dans le système 6.0 ?
- Parce que les Beatles ne veulent pas...

Pas besoin de NeXT

Lisez bien tout ce que Jean-Michel a écrit ci-dessus sur le DSP56001 de Motorola. Vous pourrez bientôt avoir cette magnifique puce sur votre GS: New Concepts prépare une carte 56001 pour le GS. Hypermédia oblige...

Patch pour Babar

Yvan Kœnig nous communique que pour rendre le Tool.Setup du système 5.04 français compatible avec le programme de J.P. Charpentier publié dans notre numéro 6, il suffit de faire le patch suivant:
à l'octet $1B8, mettre 80 13.

Pour les matheux

Seven Hills Software prépare un éditeur d'équations, qui permettra d'inclure vos hiéroglyphes dans AppleWorks-GS, Graphic Writer III, etc, et de les imprimer proprement.

VGA GS

New Concepts prépare une carte vidéo VGA pour le GS. Il faudra bien entendu un moniteur de compatible IBM.

Introduction à la programmation de l'Apple II GS :

Initiation aux ressources, 2º partie :

Resource Manager et la gestion des ressources

Jean Destelle

Cet article, comme ceux de la même série, est destiné à démystifier la "Toolbox" du GS. Dans notre premier article (ToolBox-Mag 5 p 42), nous avons vu à quoi servent les ressources, sans approfondir la façon précise dont elles sont traitées par la Toolbox du GS. Nous allons maintenant rentrer un peu plus dans le détail, et faire connaissance d'une manière approfondie avec Resource Manager (ResMgr). Auparavant, rappelons rapidement quelques notions essentielles.

1. Les nouveaux fichiers

GS/OS a introduit une nouvelle forme de fichier de type "extended", tout-à-fait analogue au type standard de fichier du Mac. Ces fichiers sont en fait doubles, et comportent deux "branches" ("forks"): la branche *données* (data fork), qui est rigoureusement identique aux fichiers précédemment utilisés, et qui peut contenir n'importe quoi, sous n'importe quelle forme, en fonction du type de fichier choisi; et la branche *ressources* (resource fork), qui est structurée d'une manière spéciale, et contient uniquement des ressources, et les moyens de ranger et de retrouver ces ressources. Cette branche du fichier est entièrement gérée par ResMgr qui court-circuite GS/OS d'une manière très efficace.

Le fait qu'un fichier soit "étendu" ou non ne peut être déterminé qu'à sa création, et ne peut pas être modifié ensuite. Le paramètre "storage-type" d'un tel fichier est $05, alors qu'il est de $01 ou $02 pour les fichiers "standard". Tout nouveau fichier peut sans inconvénient être créé sous cette forme, même si on laisse la branche de ressources vide.

2. Les ressources

2.1. Qu'est-ce qu'une ressource?

Une *ressource* est une ensemble de données regroupées dans un format défini: paramètres d'un menu, d'un dialogue, d'une fenêtre, texte d'une fenêtre d'alerte, police de caractères, icône, segment de programme, ou tout autre chose. Le format des données peut être quelconque. Il n'est pas connu de ResMgr. C'est l'application qui le détermine. Il existe quelques formats "standard" employés par les outils du GS.

Chaque ressource est définie par son *type* (numéro d'ordre donné à son format) et par son *ID* (numéro d'identification d'une ressource de type donné). Ces deux éléments définissent complètement la ressource pour ResMgr, et lui permettent de la retrouver et de la manipuler sans avoir besoin d'en connaître le format.

2.2. Types de ressources

Le type de ressource ("res.type") est un nombre entier (deux octets). On l'écrit en général sous sa forme hexa. Apple définit les types suivants:

$0000	Type non valide (ne pas employer)
$0001-$7FFF	Types autorisés pour les applications
$8000-$FFFF	Types réservés pour le système.

2.2.1. Types standard

Parmi les types réservés, Apple a défini un petit nombre de types que nous appelons *standard*, qui sont utilisables par les nouveaux outils du système 5.0. Ces types s'étagent entre $8001 et $8030. Ce sont les types que vous rencontrerez le plus souvent en employant l'analyseur de ResDoctor. ToolBox-Mag vous en a déjà présenté de nombreux exemples. Pour une liste des types de ressources définis par Apple, voyez ToolBox-Mag 6 page 27.

Pour connaître le contenu d'une ressource, il faut disposer d'une application qui puisse les analyser, c'est-à-dire décomposer (ou "désassembler") l'ensemble des données de la ressource en ses divers composants et leur donner le nom de paramètre qui les désigne. Des éditeurs de ressources comme *Derez, Genesys,* ou *ResDoctor* peuvent faire ce travail. Les ressources dont le format est inconnu ne peuvent évidemment pas être analysées.

2.2.2. Types non standard

Ils sont laissés à l'initiative du programmeur. ResDoctor se réserve d'utiliser pour l'application les numéros de types compris entre $7800 et $78FF. Le type $7800 est employé pour des ressources spéciales contenant les spécifications de format d'autres ressources, ce qui permet la création d'autant de types nouveaux analysables qu'on le désire.

2.2.3. Noms des types

Les noms donnés ici aux types de ressources sont ceux qui sont employés dans les notices de référence, et les langages de programmation les plus utilisés, mais ne sont pas utilisés par ResMgr. Dans le Mac, les types de ressources sont désignés par des mots de quatre lettres et non des numéros. La facilité d'employer des 'noms de ressources' du même genre ▶

est offerte par le ResMgr du GS, mais leur définition et leur emploi sont laissés à l'initiative de l'application.

2.3. Les *ID* des ressources

2.3.1 Attention à la confusion

Nous emploierons dans ce qui suit le terme *ID* pour désigner le *Resource ID*, bien que cela puisse prêter parfois à confusion. En effet, parmi les paramètres d'une ressource, il peut se trouver un paramètre désigné par le mot *ID*, comme par exemple le numéro d'identification d'un Item de menu ou d'un contrôle dans un dialogue. Il faut bien se garder de confondre cet *ID* avec celui d'une ressource. C'est le numéro sous lequel l'application désigne l'objet en question, et non pas le numéro de la ressource. Parfois, les deux numéros seront identiques, mais pas forcément.

2.3.2. L'*ID*

L'*ID* d'une ressource *(Res.ID)* est un entier long (quatre octets). C'est un numéro d'ordre quelconque donné par l'application à la ressource. Toutefois ce numéro doit être unique, pour le type de ressource, dans le fichier de ressources concerné, et il doit être choisi dans la plage autorisée (1 à $8000000)
En général l'*ID* se rapporte à un objet dont la désignation se fait plutôt dans les sources de programmation en système décimal. Les *ID* de ressources du GS paraissent donc le plus souvent devoir être manipulés par le programmeur sous forme décimale. C'est ainsi que les traite l'analyseur de *ResDoctor*.

2.3.3. Choix de l'ID

La fonction *UniqueResourceID* de ResMgr peut fournir un *ID* disponible pour un type donné. Cependant, la détermination de l'*ID* doit tenir compte de plusieurs facteurs, dépendant de l'emploi que veut en faire le programmeur.
Dans le Mac, les *ID* des ressources suivent une règle stricte: deux octets désignent la ressource (principale) à laquelle la nouvelle ressource appartient. Les deux autres octets désignant un numéro d'ordre parmi les ressources de ce type appartenant à la ressource principale. Il existe des fonctions permettant de trouver directement ainsi toutes les ressources affiliées à la même ressource principale. Ce système n'existe pas (à cette date) pour le GS. Le programmeur pourra donc s'inspirer éventuellement de ce système (mais ce n'est pas obligatoire) pour obtenir un classement des ressources en fonction de leur appartenance, dans l'index de la res-map, et dans les listings.
Il devra également penser que *plusieurs fichiers de ressources peuvent être gérés simultanément par ResMgr:* il suffit que l'application les ait ouverts. Dans ce cas, ResMgr cherche une ressource donnée par son type et son *ID*, dans *tous* les fichiers ouverts (s'il ne la trouve pas tout de suite). Il pourra donc y avoir confusion dans certains cas entre deux ressources de même type et de même *ID* appartenant à deux fichiers différents. (ResMgr prendra la première trouvée. Voir ci-dessous: mode de recherche).

2.4 L'attribut d'une ressource

A chaque ressource est associé un *flag d'attributs* sur deux octets, non contenu dans la ressource, mais présent dans le *reference-record* de la ressource concernée, dans l'index du répertoire (voir ci-dessous: structure du fichier-ressources). Les fonctions *GetResAttr* et *SetResAttr* de ResMgr permettent de modifier directement ce *flag*, même si la ressource n'est pas chargée. Nous désignerons dans ce qui suit ce *flag* par le nom *Attribut*. Son utilité est tout-à-fait comparable à celle de l'attribut utilisé par Memory Manager pour les handles. Il sert à définir un certain nombre d'options concernant la façon dont ResMgr doit gérer la ressource en mémoire. Vous en trouverez le tableau dans ToolBox-Mag 3 p39 (article de B. Fournier).

3. La gestion des ressources

3.1. Structure du fichier-ressources

Nous désignerons dans ce qui suit par le terme *fichier-ressources,* aussi bien que par la désignation anglaise d'origine *res-fork,* la branche ressources d'un fichier de type *extended,* quoique ce ne soit pas à proprement parler un fichier au plein sens du terme, mais seulement une partie annexe d'un fichier. Plusieurs fonctions de Resource Manager permettent de manipuler ces fichiers, de les créer, les ouvrir et les fermer, d'aller y chercher des ressources, les modifier, en ajouter, en supprimer, etc. Une application n'a pas en principe à se soucier de la manière dont le fichier-ressources est constitué. Cependant, il est bon de comprendre sa structure.
Le fichier-ressources est composé essentiellement des ressources elles-mêmes *(resource data)* et d'un répertoire *(Resource-Map).* La *Resource-Map* ou *ResMap* est tenue à jour de façon permanente par ResMgr, et comporte toutes les données utiles au rangement et à la recherche des ressources dans le fichier. Lorsqu'on ouvre un fichier-ressources, ResMgr charge en mémoire le répertoire. Les ressources sont écrites là où il y a de la place dans le fichier: seul le premier bloc de tête *(header)* occupe un emplacement fixe. Il comporte les renseignements suivants (l'origine des décalages [**Déc**]se situe au premier octet de la res-fork):

Format du *Header*

Déc	Format	Désignation	Commentaires
$00	Long	rFileVersion	Version: GS:0 ; Mac: >127;
$04	Long	rFileToMap	Offset de la res-map;
$08	Long	rFileMapSize	Taille, en octets, de la map;
$0C	128 octets	rFileMemo	Disponible pour l'application.

La *ResMap* comporte elle-même un *header* suivi de deux tableaux: la liste des emplacements libres du fichier *(Map Free List)* et le répertoire des ressources *(Map Index)*.

▶

Format de la *Map*

Déc	Format	Désignation	Commentaires
$00	Long	mapNext	Handle vers fichier suivant;
$04	Word	mapFlag	Indique si la map a été modifiée;
$06	Long	mapOffset	identique à rFileToMap du header;
$0A	Long	mapSize	Taille actuelle de la map;
$0E	Word	mapToIndex	Décalage début map / index;
$10	Word	mapFileNum	No du fichier donné par GS/OS;
$12	Word	mapID	ID du fichier donné par ResMgr;
$14	Long	mapIndexSize	Nb total de références dans l'Index;
$18	Long	maxIndexUsed	Nb de références utilisées;
$1C	Word	mapFreeListSize	Nb de blocs libres utilisés;
$20	Array..	mapFreeList	Tableau des blocs libres;
$xx	Array..	mapIndex	Tableau de "references-records".

Blocs libres: dans le tableau *MapFreeList*, chaque emplacement libre est représenté par un *record* de deux paramètres:

Format d'un bloc libre

Déc	Format	Désignation	Commentaires
$00	Long	blokOffset	Position du bloc libre;
$004	Long	blkSize	Taille du bloc libre.

Le tableau le plus important est le répertoire (*Index*) qui fournit les adresses des diverses ressources, ainsi que quelques données les concernant.

Format d'un record *Resource Reference* dans l'index

Déc	Format	Désignation	Commentaires
$00	Word	resType	Type de la ressource; (1)
$02	Long	resID	ID de la ressource;
$06	Long	resOffset	Offset de res. (0=début res-fork);
$0A	Word	resAttr	Attribut (voir ci-dessous);
$0C	Long	resSize	Taille de la res. dans le disque; (2)
$10	Long	resHandle	handle sur res. en mém.; (3)

Remarques:

(1) ResMgr range les références de l'index dans l'ordre croissant des types, puis dans chaque type, dans l'ordre croissant des *ID*. En cas de besoin, il est facile de retrouver le record de référence d'une ressource en parcourant le tableau d'index au moyen d'un éditeur de blocs.

(2) La taille en mémoire d'une ressource n'est pas forcément la même que dans le disque pour deux raisons: d'une part, l'application a pu modifier la ressource après son chargement; d'autre part, il peut exister une "fonction de conversion" qui, automatiquement, transforme la ressource au moment de son chargement (par exemple: décompression d'une image). Voir l'article de B. Fournier dans ce numéro.

(3) Ce paramètre n'a évidemment de valeur qu'en mémoire.

3.2. Gestion des fichiers-ressources

3.2.1. Initialisation de Resource Manager

ResMgr est automatiquement chargé en mémoire dès le début de l'application. Il se réserve les pages de mémoire dont il a besoin. Lors de sa fermeture, au moment de quitter l'application, il referme tous les fichiers-ressources qui sont restés ouverts.

3.2.2. Manipulations du fichier

Les fichiers de type "étendu" peuvent être manipulés par GS/OS: on peut ouvrir la res-fork seule d'un tel fichier avec l'une des fonctions de GS/OS. Cet outil allouera au fichier ouvert un numéro d'identification (*ID* 'GS/OS') nécessaire pour sa fermeture. Toutefois, si on désire travailler sur les ressources de ce fichier, et pas seulement le recopier, ce procédé n'est pas bon. Il faut utiliser les fonctions de ResMgr:
OpenResourceFile ouvre le fichier concerné, et charge la map en mémoire. Il fournit un numéro d'identification *(ResFileID)* qui lui est propre et servira de repère par la suite. Cette fonction peut concerner un fichier "étendu" quelconque, mais ne s'occupera pour l'instant que de sa res-fork. Elle ignorera la data-fork. Une fois le fichier ouvert, ses ressources deviennent accessibles par ResMgr, et il devient le premier fichier pris en compte dans la recherche si on veut charger une ressource.

3.2.3. La séquence de recherche

ResMgr peut ouvrir successivement plusieurs fichiers-ressources, et il est ainsi possible de disposer à la fois d'éléments provenant de plusieurs fichiers différents. Le fichier-ressources du système *(Sys.Resources)* est ouvert dès l'initialisation, et reste toujours ouvert.
Lorsque ResMgr est appelé pour charger une ressource, il effectue une recherche dans les *res-maps* qu'il a en mémoire, jusqu'à y trouver une ressource ayant le type et l'*ID* recherchés. La séquence de recherche commence par le dernier fichier ouvert, puis l'avant-dernier, etc, et se termine par le fichier *Sys.Resources*. On peut modifier l'ordre de recherche en utilisant la fonction *SetCurResourceFile* qui met en première ligne le fichier indiqué, et définir le nombre des fichiers explorés grâce à la fonction *SetResourceFileDepth* qui retourne le nombre de fichiers explorés avant son intervention.

3.2.4. Enregistrement

La fonction *UpDateResourceFile* permet d'enregistrer sur disque les modifications effectuées en mémoire sur l'ensemble des ressources d'un fichier. *CloseResourceFile* enregistre les modifications intervenues depuis l'ouverture, réécrit la nouvelle ResMap sur disque et referme le fichier.

4. La gestion des ressources

4.1. Chargement d'une ressource

Lorsque le fichier-ressources a été ouvert, ResMgr peut retrouver rapidement toute ressource contenue dans ce fichier ou dans l'un des autres ouverts, grâce à la fonction *LoadResource*, qui admet comme arguments le type et l'*ID* de la ressource, et retourne un handle sur le bloc de mémoire contenant la ressource. Si la ressource ne se trouve pas déjà en ▶

mémoire, ResMgr la recherche, lui alloue un handle, et retourne ce handle à votre programme.

4.2. Utilisation de la ressource

Vous pourrez alors utiliser les données contenues dans cette ressource, et même manipuler la ressource, la modifier, changer éventuellement sa longueur. Si vous désirez que les changements soient reportés dans le disque, vous devrez utiliser la fonction *MarkResourceChange*, et la ressource sera enregistrée à la prochaine occasion.

L'utilisation principale de la ressource sera toutefois le plus souvent une utilisation statique, c'est-à-dire sans modification. Par exemple, l'emploi d'une chaîne Pascal pour un affichage, ou bien l'utilisation d'une ressource comme "template" par l'un des outils du GS, ou comme un "record" (collection de données de format variés classées dans un ordre défini).

Il n'entre pas dans le cadre de cet article d'exposer toutes les possibilités d'emploi, qui sont innombrables. Tout programmeur expérimenté a souvent été confronté à la multiplicité des petits fichiers auxiliaires nécessités par une application un peu complexe. Réduire, quand c'est possible ces petits fichiers à l'état de ressources rendra leur accès infiniment plus pratique et plus rapide que par GS/OS.

Resource Manager est en quelque sorte un "magasinier" rapide et efficace, qui étiquette votre ressource avec le numéro que vous avez choisi, la range, et vous la redonnera instantanément quand vous en aurez besoin. Vous n'avez pas à vous préoccuper du mode de stockage. Seul vous intéresse le contenu du paquet que vous lui avez confié. Ce contenu, vous le définissez en créant le "format" de votre ressource, ou bien vous utilisez un format standard. Mais cela n'est pas l'affaire de ResMgr. C'est votre problème.

La ressource étant repérée en mémoire par son handle, votre application peut en utiliser les données. Vous pouvez les lire, les recopier et les modifier. Attention toutefois à ne pas considérer ce handle comme un handle ordinaire ! Il appartient à ResMgr, qui veille jalousement sur lui et a seul le droit d'en disposer. Si vous voulez l'utiliser comme un handle ordinaire, il faut le "détacher" de ResMgr en utilisant la fonction *DetachResource*. La ressource reste alors en mémoire dans le même bloc, mais ResMgr ne peut plus s'en occuper.

4.3. Création d'une ressource

ResMgr n'a pas de fonction qui crée directement des ressources. Il faut un "Compilateur" ou un "Editeur" de Ressources, comme ResDoctor.

Vous pouvez également inclure dans votre programme des procédures qui ajoutent des ressources nouvelles à un fichier existant, (ou bien en suppriment). Les fonctions *AddResource* et *RemoveResource* de ResMgr sont là pour cela.

4.4. Autres fonctions de ResMgr

Nous ne pouvons étudier ici dans le détail toutes les fonctions de ResMgr. Nous nous limiterons à citer quelques outils parmi les plus utiles.

• *GetResourceAttr* vous permet de connaître l'attribut d'une ressource, et *SetResourceAttr* de le modifier. La modification effectuée ne sera prise en compte qu'à la prochaine utilisation de la ressource par *LoadResource*.

• *GetResourceSize* vous donne la taille de la ressource dans le disque. Pour avoir sa taille en mémoire, appelez la fonction *GetHandleSize* avec comme argument le *ResHandle* fourni par *LoadResource*. Il peut y avoir une différence importante entre ces deux données, si la ressource est soumise à une fonction de conversion pendant le chargement.

• *SetResourceID* pourra modifier l'*ID* de la ressource. A employer avec précaution !

4.5. Interdépendance des ressources

Il est très fréquent, notamment dans le cas des ressources standard, qu'une ressource d'un certain type fasse référence, parmi les données qu'elle contient, à des ressources d'autres types. La création des fichiers ressources, de même que leur modification, doit tenir compte de ces relations. C'est pourquoi la gestion des ressources demande de la part du programmeur beaucoup d'ordre et de méthode. Voyrz l'article de B. Fournier dans ToolBox-Mag 3 p 39.

4.6. Les erreurs fréquentes

L'emploi de ressources conduit parfois à des blocages de programmes, principalement lorsque ResMgr ne dispose pas des ressources recherchées. Quand il s'agit de blocs de paramètres utilisés par les outils du GS, la fonction en cours d'exécution s'arrête, et, si on a pris la précaution de prévoir un message d'erreur, un numéro d'erreur $1E06 risque d'apparaître.

Il peut se faire simplement que le préfixe utilisé pour retrouver la ressource en question dans le disque ait été perdu, pour une raison ou une autre. Veillez donc à ce que votre application revienne toujours au préfixe qui lui a été attribué.

Il pourra vous arriver aussi des accidents fortuits qui auront supprimé ou abîmé l'une de vos ressources dans le disque. Comme ResMgr réécrit souvent dans le disque ce qui a été modifié en mémoire, il n'est pas du tout impossible que cela arrive, surtout quand vous expérimentez de nouveaux programmes qui ne sont pas au point. Pensez donc à vérifier vos ressources si un ennui de ce genre survient.

La meilleure précaution est de sauvegarder souvent les fichiers de ressources, et de conserver un listing imprimé de l'analyse des ressources, obtenue avec un analyseur comme celui de ResDoctor. Cette application a été conçue pour vous assister dans ce cas.

Nous avons passé en revue, dans cet article, les points les plus importants concernant la gestion des ressources par ResMgr. Nous disposons maintenant des connaissances essentielles pour aborder le côté pratique de leur utilisation. ■

GS News

Eric Weyland

Ce qu'il faut savoir

Roger Wagner et l'avenir de l'Apple IIGS

Dans un récent article paru dans Apple 2000 (numéro de juin 1991), Roger Wagner (éditeur entre autres de HyperStudio, Merlin 16, Macromate...) fait le point sur l'avenir de l'Apple IIGS.
Pour lui, le point fort de l'Apple IIGS est sans contestation possible tout ce qui touche à l'hypermédia (HyperStudio mais aussi HyperCard couleur). Si les années 80 ont vu le développement des applications graphiques et de PAO, les années 90 vont être le début de l'essor de l'hypermédia. La machine idéale doit posséder des capacités couleurs puissantes, un son excellent, et doit avoir la possibilité d'être connectée à des lecteurs de CD-Rom, des magnétoscopes... La machine doit aussi être simple d'emploi; l'Apple IIGS est l'un des ordinateurs les mieux adaptés, que ce soit au niveau technique ou au plan financier où il est compétitif pour les applications hypermédia.

Une version 3.0 d'HyperStudio devrait voir le jour le 15 Septembre. Elle prendrait en compte les fonctionnalités du système 6.0, les images 3200 couleurs, elle permettrait les menus détachables, et aurait enfin un langage de programmation à la hauteur d'HyperTalk. Plus de 10000 copies d'HyperStudio ont été vendues de par le monde.

Macs à bas prix : ça eût payé...

La vente en masse des Macintosh Classic et LC ne satisfait pas outre mesure les concessionnaires Apple. En effet leur taux de marge a été réduit par Apple, faisant que leur profit s'en trouve aussi diminué. ToolBox Mag leur conseille de vendre des Apple IIGS. Le prix de cet ordinateur n'ayant pas changé, il est plus rentable pour un concessionnaire de vendre un GS plutôt qu'un Mac Classic ou LC.

Système 6.0 (Apple Computer)

Apple a rendu publiques à la KansasFest les caractéristiques du système 6.0 de l'Apple IIGS, dont la sortie est prévue pour fin décembre 1991. (en ce qui concerne la date de disponibilité de la version française, consulter l'"Apple II Service Team' d'Apple France: c'est son travail).
Il comprend un Prodos 8 version 2.0 qui permet plus de deux volumes par slot (le total de 14 volumes de 32 Mégas reste inchangé, mais ça nous suffira pour le moment).
Il inclura trois nouveaux FSTs (voyez ToolBox-Mag 6 page 33 sur la notion de FST). On pourra donc, sous GS/OS, lire les volumes Dos 3.3 et Pascal-UCSD, mais surtout lire et écrire des volumes HFS/Macintosh. Non seulement l'échange avec le Mac sera facilité, mais surtout GS/OS pourra désormais gérer concrètement des volumes de plusieurs GigaOctets (des milliers de MégaOctets!).
Côté drivers, le 6.0 comprendra un driver de RamDisk qui accélérera encore les choses, un driver SCSI pour les scanners, un driver SCSI pour les sauvegardes sur bande magnétique, un driver de StyleWriter (mais sans TrueType).
Une application de sauvegarde universelle, l'Archiveur, permettra de faire un back-up de tout périphérique sur un autre périphérique (par fichiers ou par blocs).
Les utilitaires pour handicapés (CloseView, Video Keyboard, Sticky Keys et Mouse Keys), actuellement diffusés par l'APDA, seront diffusés avec le système.
L'installeur deviendra vraiment un installeur universel, qui sera utilisé par ▶

tous les logiciels que vous achèterez (dont sans doute ToolBox-Mag).
Le nouveau Finder (vous connaissez son auteur) ira plus vite dans tout un ensemble d'opérations, facilitera les opérations de marquèterie de fenêtres, aura ses icônes en ressources, permettra de mettre des commentaires en ressources dans les applications, dialoguera encore mieux avec les accessoires, Inits et CDevs, grâce à des "hooks" internes, et connaîtra beaucoup d'autres innovations. Il pourra attribuer les fichiers aux applications grâce à des ressources Bundle, comme sur Mac.
Le 6.0 contiendra également un bon paquet d'innovations dans la ToolBox: un nouvel outil *Media Control* permettra de s'interfacer avec les lecteurs de compact-disques, magnétoscopes, etc.
On pourra avoir des icônes dans les menus, des équivalents-clavier pour les boutons dans un Contrôle d'Alerte, le Resource Manager gèrera mieux les noms de ressources, etc.
Le 6.0 sera livré sur cinq disquettes (!), et si vous ne savez toujours pas qu'il faut à votre GS, comme à tous les ordinateurs d'aujourd'hui, de la mémoire et un disque dur confortable, vous allez bientôt en avoir la preuve.

Compuserve en France

Une version limitée du serveur américain est disponible via le 3617 code Compu. On y trouve des informations en anglais sur le monde de l'Apple II ainsi que de très nombreux programmes à télécharger.

L'accord Apple/IBM

Puisque vous avez été plusieurs à nous demander notre avis sur l'accord Apple/IBM, disons-en un mot. En-dehors de choses qui nous indiffèrent, comme la nique à Microsoft, cet accord ne fait que confirmer une des tendances majeures que chacun constate dans l'industrie: l'uniformisation dans des machines de pur travail, où peu importe le matériel, où l'utilisateur souffre sur XPress et Lotus 1/2/3 dans un environnement graphique-souris, tandis que les usines de programmation reçoivent trente kilos de nouvelle documentation développeurs chaque semaine.
C'est déjà ce qui se passe: un compatible 386 sous Windows 3 ou un Mac 68030 sous système 7 sont pratiquement interchangeables, pour l'utilisateur aussi bien que pour le programmeur. Ce qui caractérise cette informatique-là, dite "professionnelle", c'est qu'elle n'a strictement aucun intérêt, elle est mortellement ennuyeuse.
Heureusement, il y aura toujours des Wozniak et des Jobs, des individus qui voudront faire de l'informatique intéressante, pour le plaisir. C'est d'eux, comme d'habitude, que viendra la nouveauté. C'est eux qui périmeront tous les accords Apple/IBM d'un seul coup d'innovation. Les colonnes de ToolBox-Mag leur seront toujours ouvertes.

Le hard

GS Memory (CV Technology)

Cette carte d'extension mémoire 4 Mégas accepte la carte Apple ou d'autres cartes 4 Mégas (mais pas les cartes Æ) dans son dos: vos anciennes cartes ne sont donc pas perdues. Seuls les premiers 4 Mégas de la carte sont accessibles en DMA, mais ça suffit bien.

TransWarp GS (Æ) : cache 32 Ko

La carte accélératrice TransWarp GS d'Applied Engineering peut maintenant recevoir une option portant sa mémoire cache à 32 Ko. Cela accroît sa vitesse de 20 à 50%. Les mises à jour devraient bientôt être disponibles.

Le futur GS (Apple Computer)

A la Kansas Fest, on a pu apprendre quelle nouvelle version du GS Apple prépare. Ce sera en gros le même modèle que le modèle actuel (pas d'accélérateur ni de nouvelle vidéo), mais la configuration de base aura deux Mégas de Ram et un disque dur 40 Mo interne, et sera livrée avec HyperCard.

EtherNet sur GS (Apple Computer)

AppleTalk, c'est lent. EtherNet, ça va plus vite. C'est le nouveau standard pour les réseaux chez Apple, comme François Hermellin vous l'a déjà expliqué.
En conséquence, Apple a annoncé à la Kansas Fest qu'il avait en préparation une carte EtherNet pour tous les Apple II.

SuperDrive 1600K (Apple Computer)

Le standard actuel pour les lecteurs de disquettes chez Apple est le lecteur 3,5 Sony 1600K. En consé- ▶

quence, Apple a également annoncé un ensemble carte d'interface/lecteur Super-Drive pour les Apple II.
Il me semblerait tout-à-fait légitime que les utilisateurs de Photonix II puissent acheter une mise à jour pour lecteur 1600k quand celui-ci sera disponible. Chiche, Olivier ?

Liberty (Micol Systems)

En attendant le SuperDrive Apple, il existe déjà un lecteur 1600 ko sur lequel on peut booter en P8 (à la différence du lecteur Æ), et sur lequel on peut lire et écrire les disquettes MS/Dos (comme sur le lecteur d'Æ): il d'une carte contrôleur et d'un lecteur produits par Micol Systems, les Canadiens qui font déjà Micol Basic.
Le logiciel pour lire et écrire en MS/Dos est fourni avec. Comme le 6.0 ne comprendra pas, sauf contre-ordre, de FST MS/Dos, voilà un bon filon pour Micol.

Pegasus (Econ Technologies)

Un disque dur interne de 200 Mégas (ou de 50, ou 100), voilà une sérieuse concurrence au Vulcan d'Æ. Surtout quand on sait qu'il s'agit d'un lecteur SCSI, qui marche avec la SCSI/DMA d'Apple comme avec la RamFast SCSI.
On peut même acheter une version en kit, sans le disque dur, où l'on ajoute soi-même sa "gamelle" de disque dur 3,5.
Le tout est fourni avec des utilitaires de formatage, de récupération de disques abîmés ou de fichiers détruits, de backup, et même un optimiseur!

Le soft: les jeux

Sluggo (Bill Heinemann)

Quand on s'appelle Bill Heinemann, qu'on a écrit entre autres Bard's Tale GS et Crystal Quest GS avec Merlin 16, et qu'on entend parler de la console de jeux SuperNintendo, dont le processeur est un 65816 comme sur le GS, que fait-on ?
Eh bien, on fait un ensemble matériel et logiciel pour programmer cette console et y télécharger ses programmes, de façon à les tester (c'est-à-dire à jouer, n'est-ce pas). Bill appelle cet ensemble Sluggo, et ne me demandez pas pourquoi.

Geo Quiz (PC Globe)

Pourquoi donc avais-je cette impression de déjà vu face aux écrans de l'excellent jeu de connaissances géographiques GéoQuiz ? J'ai fouillé vainement dans mes souvenirs, jusqu'à ce que j'aie l'idée de regarder le nom des auteurs: Xavier Schott, Jean-Pierre Curcio, Nathalie Mérino, Luc Sérard, ça ne vous rappelle rien ?
Eh oui, notre bon vieux *Atlas 2000* semble avoir mis quatre ans pour traverser l'Atlantique et trouver un éditeur aux USA. Auteurs français, ne désespérez pas...

Jungle Safari (Orange Cherry Software)

Safari éducatif dans le jungle africaine. Excellents graphiques, jeu très intéressant où l'on apprend en s'amusant.
La documentation comprenant des instructions parfaitement claires pour l'installation sur le dur, j'en profite pour dire un mot de polémique à nos confrères de GS+: quand donc cesserez-vous de dire que ces logiciels sont longs à charger quand on les boote depuis leur disquette 3,5 ?
On ne boote jamais une disquette sur GS, comme d'ailleurs sur Mac, IBM ou n'importe quel ordinateur d'aujourd'hui. On installe le logiciel sur le dur, et on l'y lance sous GS/OS. Amis de GS+, *avez-vous songé à tester ce programme sur disquette 5,25 ?* Eh bien, c'est encore plus lent!
Ce genre de critique rétro des nostalgiques du lecteur de cassettes n'aura plus lieu d'être bientôt: avec le 6.0, Orange Cherry n'aura plus besoin de mettre un vieux P16 sur la disquette. On y mettra le logiciel plus des fichiers pour l'Installeur, et les dinosaures seront bien forcés d'apprendre qu'un disque dur 40 Mégas coûte moins cher qu'un lecteur de disquettes.

Les lâcheurs

Wings of Fury, Ancient Land of Ys, etc, vous ne pouvez plus les acheter: Broderbund Software vient d'arrêter toutes ses activités (édition, distribution) en ce qui concerne les logiciels de jeux.
The Immortal, ChessMaster 2100, Zany Golf, vous ne pouvez plus les acheter non plus, Electronics Arts cessant toute activité pour les jeux Apple II.
On comprend pourquoi Bill Heinemann se met à programmer pour SuperNintendo.
Renseignements pris, Toolbox continue toujours à commercialiser Gate...

▶

Le soft: les utilitaires

Claris : vraiment pro ?

Vous vous rappelez sans doute qu'à l'Apple-Expo, les employés de Claris répondaient qu'AppleWorks-GS n'était pas un logiciel Claris. Eh bien, quelqu'un s'est aperçu que si, et la dernière version de Mac Write II que nous ayons vue (celle qui n'est pas encore "Pro") est capable d'importer des fichiers traitement de texte d'AppleWorks-GS.

Il ne manque plus qu'à FileMaker Pro d'importer les fichiers Base de données d'AppleWorks-GS, à Mac Draw d'importer les fichiers graphiques AppleWorks-GS, à Resolve d'importer les fichiers tableur, pour que nous cessions nos fous-rires quand Claris s'affuble du titre de "Pro".

Enfin, il faudrait encore que Claris cesse de se ridiculiser à la Kansas Fest en y montrant des Macs, ou des GS en panne...

Talking Tools (ByteWorks)

Bien entendu, c'est *deux jours* après l'envoi de ToolBox-Mag 6, contenant l'étude de Jacques Liautard sur les différentes manières de faire parler le GS, que nous est arrivé le nouvel ensemble logiciel de ByteWorks appelé Talking Tools pour faire parler le GS.

C'est toujours comme ça. Ils le font exprès. Eh bien nous en parlerons quand même dans un prochain ToolBox-Mag...

Le retour de l'Integer Basic (ByteWorks)

Depuis qu'Apple France a mis, comme on sait, le paquet sur l'Integer Basic et la compatibilité Apple II+ pour Mac LC, l'Integer Basic revient à la mode.

ByteWorks vient donc de sortir un compilateur Orca pour l'Integer Basic. Attention, c'est un compilateur 16 bits, il ne tourne pas sur II+ ni sur Mac LC. Compilez "Eliza" (vous vous rappelez ?), et vous aurez un Eliza GS/OS!

Non, ce n'est pas un poisson d'Avril. Il y a *réellement* une carte compatible II+ pour Mac LC et un compilateur Orca/Integer Basic pour Apple II GS! En prime (et ça, c'est intéressant), avec le second vous aurez le source complet d'un compilateur Orca (ainsi d'ailleurs qu'une disquette pleine de programmes en Integer...).

Orca/M 2.0 (ByteWorks)

Toujours ByteWorks, avec une annonce moins rétro cette fois-ci. Il semble bien qu'il n'y aura jamais d'APW 2.0: en tout cas, Orca 2.0, l'assembleur de ByteWorks, sera livré avec le débugger GS Bug comme avec Rez/Derez. Plus un nouvel éditeur (il était attendu), et la compatibilité avec le système 6.0, bien sûr.

SDE, un nouveau shell (SDA Software)

ByteWorks ferait bien de se dépêcher: ceux qui avaient écrit l'excellent MaxEdit, dont B.Fournier vous a dit tant de bien, viennent de compléter l'essai avec un environnement de programmation complet, un remplaçant au shell d'APW-Orca.

Ce "Software Development environment" (SDE), édité par SEA Software, est bien supérieur, dans tous les compartiments du jeu, à APW-Orca. Il concilie l'aspect unixien cher aux programmeurs en général avec la vitesse d'assemblage de Merlin, chère à ceux qu'Olivier Goguel appelle les "vrais programmeurs".

Revue dans un prochain numéro de ToolBox-Mag.

TGE Library Disk 1 (Roger Wagner Publ)

A partir de la version 4.20 de The Graphic Exchange (TGE), il est possible de construire de nouveaux Toolsets pour que le programme prenne en compte d'autres formats graphiques que ceux prévus à l'origine. En copiant un nouveau fichier Toolset dans le sous-catalogue /TGE.TOOLS, TGE peut travailler avec un format graphique supplémentaire.

Ce disque Library numéro 1 contient des Toolsets pour les formats graphiques suivants : Works of Art de Springboard Publisher, les bordures de PrintShop et de PrintShop GS, les données Superprint (stationery, backgrounds, posters et frames), les dessins Mac Paint, les fichiers GIF, les fichiers PCX (IBM), et les images en 3200 couleurs.

Un source d'exemple d'un Toolset est explicité, permettant ainsi d'écrire vous même un Toolset.

En attendant le 6.0

La série de NDA tant attendue DeskPack Volume 2, de SSSI (voir ToolBox Mag numéro 5) n'est pas encore disponible. La sortie imminente du système 6.0 en est la cause, il faut faire les tests de compatibilité.

Même raison pour Genesys 2.0, toujours de SSSI. Attention quand même, ▶

amis de SSSI, après le 6 il y aura le 7, et on risque de ne jamais sortir son logiciel, à force de le réviser pour le système suivant.

GNO (Procyon Inc) : Unix sur Apple IIGS

En attendant de craquer pour le NeXt que Jean-Michel Vallat vous vante dans ce numéro, vous pouvez déjà goûter aux joies d'Unix (la ligne de commandes à la APW, mais aussi le multitâches) sur votre Apple IIGS. Développé sous Orca, GNO se présente comme un système multitâches sous GS/OS. GNO switche entre les différents programmes 60 fois par seconde. GNO est livré avec un shell appelé GSH qui en fait un véritable environnement UNIX.

SuperWorks

"Votre Apple II est une machine obsolète. Nous en avons une nouvelle à vous vendre. Elle vous permettra de reprendre tous vos fichiers AppleWorks, et des tas d'autre chose. Qu'attendez-vous donc pour changer de machine ?".

Cette rengaine-là, vous la connaissez: elle vient d'Apple lui-même. Ne voilà-t-il qu'elle vient d'être prise au mot par une des entreprises piliers de l'Apple II, à savoir Resource Central ?

Oui, mais la nouvelle machine dont il est question, ce n'est pas un Mac. C'est un *compatible IBM*, et le logiciel dont il est question, qui non seulement reprend tous les fichiers AppleWorks 8, mais est *un clone complet d'AppleWorks*, en plus puissant, s'appelle SuperWorks.

En le mettant à son catalogue, Resource Central adresse un message clair à Apple, semble-t-il. Voilà ce qu'on gagne avec ce genre de rengaine.

Ceci dit, qualifier le GS d'obsolète n'est rien d'autre qu'une ânerie, quelle que soit sa provenance, et SuperWorks pour IBM, tout comme AppleWorks 8 d'ailleurs, est un logiciel largement obsolète, comparé à... AppleWorks-GS, tout simplement!

HyperCard GS 1.1

Dans les annonces d'Apple à la Kansas Fest, la version 1.1 d'HyperCard. Essentiellement des améliorations internes (de vitesse et de fiabilité), mais aussi quelques fonctions nouvelles (possibilité de programmer des fenêtres), et bien sûr la compatibilité avec le 6.0.

Cela signifie essentiellement qu'HyperCard, sur GS comme sur Mac, est un produit suivi: à chaque nouvelle version du système, à chaque nouvelle machine, une nouvelle version d'HyperCard.

Dream Graphics (DreamWorld Software)

Vous vous rappelez la magnifique démo de ce logiciel graphique 3200 couleurs ? Eh bien, ça y est, ce n'est plus un rêve mais une réalité.

Précisons: Dream Graphics n'est pas *seulement* un programme de Paint en 3200 couleurs. C'est un logiciel de dessin complet, supportant tous les modes graphiques du GS (4, 16, 256 couleurs...).

Bien entendu, il est au standard Prodos (installable sur le dur, supportant les accessoires, etc). En fait, c'est un sérieux concurrent pour Platinum...

File Detective (DataComb)

Ce nouveau NDA permet toute recherche de fichiers, avec différents critères, sur l'immense jungle de votre disque dur. Il permet aussi de chercher une chaîne à l'intérieur des fichiers, y compris dans la branche ressources.

Le soft: freeware et shareware

Excuses

Pendant que j'écris ces lignes, une vahiné vous apporte sur la plage, au bord de votre chaise-longue, un cocktail au citron vert et au lait de noix de coco. Et moi...

Eh bien moi, j'ai fait une pile avec les disquettes de freeware et de shareware, je vous en parlerai au numéro 8. Vous voudrez bien m'excuser. Attends, vahiné, j'arrive!

A lire

Script Central (Resource Central)

Pour ceux qui veulent des piles HyperCard de qualité ainsi que de bons conseils dans l'utilisation de cet indispensable logiciel hypermédia, nous conseillons vivement Script Central: la "lecture" (dit-on qu'on "lit" une pile HyperCard ? Et sinon, que dit-on ?) du numéro 1 nous a véritablement étonnés.

ToolBox-Mag souhaite se mettre, lui aussi, aux piles HyperCard. Se mesurer à ce standard-là, ça ne va pas être facile...

■

Courrier des lecteurs

***F. Lemarchand:** Je n'ai trouvé nulle part sur la disquette du numéro 4 le fichier "Wings.Quitter" dont vous parlez dans "Le monde à l'endroit". Je voudrais aussi que vous sachiez qu'il est inutile de répéter toutes les deux pages que le GS n'est pas mort: les prodigieuses réalisations qui se trouvent sur les disquettes tous les deux mois en sont la preuve suffisante!*

ToolBox-Mag : Vous avez raison sur les deux points, et merci de vos compliments. Pour le fichier Wings.Quitter, il est si petit et si facile que nous l'avons oublié: nos excuses. Il y a une solution toute simple: prenez le petit fichier appelé "Lancez.Moi" du catalogue /DIVERS de cette disquette 4, copiez-le, et renommez cette copie "Wings.Quitter": il vous fera le même usage, à savoir faire un Quit GS/OS. Simplement, il vérifiera auparavant que vous êtes bien sous 5.04.

***M.Carpentier:** Je n'ai pas été entièrement convaincu par l'article de J.Y. Bourdin sur le système 5.04 français dans le numéro 4. Je remercie, bien sûr, les "dépanneurs" qui nous permettent d'avoir ce nouveau système sans délai et à coup sûr. Mais j'ai acquis le système 5.02 chez mon concessionnaire Apple, et c'est là que je veux continuer à trouver les nouveaux systèmes, pour le GS comme pour le Mac.*

ToolBox-Mag : Rappelons que, si la diffusion (payante) des nouveaux systèmes GS et Mac est une **obligation** pour les concessionnaires Apple, ils ont aussi le **droit** de vous faire une **copie gratuite** des disquettes correspondantes. Or, le 5.04 ne consiste précisément qu'en deux disquettes.
Le monopole des concessionnaires sur les produits Apple, et le surcoût que nous payons de ce fait, ont pour excuse officielle ce qu'Apple appelle le "service supplémentaire" rendu par ses concessionnaires. Si ceux-ci refusent de diffuser les systèmes Apple, et que seuls des "dépanneurs" extérieurs acceptent de rendre ce service, il y a effectivement quelque chose qui ne va pas.
Le système 6.0 va bientôt donner l'occasion à Apple-France, à ses concessionnaires et à son "Apple II Service Team" de faire leur travail.

***R.Santelli:** Dans une carte d'une pile HyperStudio, on peut faire défiler beaucoup de texte. Mais on ne peut imprimer sur ImageWriter que l'écran du début. Comment alors communiquer copie des sujets traités aux amis?*

ToolBox-Mag : En leur donnant une disquette avec votre pile. Cela implique bien sûr qu'ils aient un ordinateur, et le même que vous.
C'est la règle du jeu de l'hypermédia: l'écran de l'ordinateur est censé remplacer le papier, les revues et les livres. L'imprimante ne sert donc qu'à faire des copies de ces écrans, pour l'utilité du programmeur. La communication avec les amis passe par l'ordinateur.
Bien entendu, cela se discute. L'expérience de ToolBox-Mag et de sa formule "*revue et disquettes*" tendrait plutôt à montrer qu'hypermédia et papier se complètent.
Quoi qu'il en soit, même s'il serait souhaitable qu'HyperStudio sache imprimer tout le texte d'une carte, un logiciel d'hypermédia n'est pas *destiné* à l'impression sur papier: il faut plutôt chercher du côté des traitements de textes et programmes de PAO.

A propos de la LQ : Certains d'entre vous rencontrent des problèmes avec le driver de LQ du 5.04. Rappelons qu'il vous faut des **fontes de taille triple** (par exemple du 36 pour imprimer en 12), mais **réellement** de taille triple pour obtenir la meilleure qualité d'impression.
N'ayant pas de LQ, nous ne pouvons pas faire ces fontes, mais seulement vous aider à les faire. Il faut utiliser le Fontasm 2.0 de S. Hadinger (Ed Mev), relire l'article de J.Y Bourdin dans ToolBox-Mag n° 1, et surtout **gâcher beaucoup de papier** pour vérifier ce que cela donne réellement. La qualité d'impression de la LQ mérite cet effort: c'est une **excellente** imprimante.
Ne désespérez pas: bientôt, avec le 6.0 et le driver de StyleWriter, tout un paquet de triplettes de fontes vérifiées vous sera offert par Apple dans le système et par... ToolBox-Mag sur ses disquettes.
Si vous avez une ImageWriter simple, ces triplettes vous serviront aussi. ■

Sommaire des numéros 1 à 6 de ToolBox-Mag

Genre	Titre	Auteur	N°	Situation
Goodies	Trois "Two-Liners"	P. R-Richard	3	Disq 03
HyperCard	Pile Bonjour	J. Liautard	6	Disq 06B
HyperCard	Pile Démo sons HyperCard	J.Y. Bourdin	6	Disq 06B
HyperCard	Premiers conseils HyperCard	J.Y. Bourdin	4	Page 47
Infos	ConvSecondes	Agent X27	3	Disq 03
Infos	Le plus grand réseau du monde	P. Pointu	4	Page 32
Infos	Le système nouveau est arrivé	J.Y. Bourdin	4	Page 04
Infos	Mac et GS	J.Y. Bourdin	3	Disq 03
Infos	Précisions sur ToolBox-Mag	J.Y. Bourdin	2	Page 19
Infos	Supplique aux utilisateurs de GS	J.Y. Bourdin	2	Page 34
Infos	Wanted	J.Y. Bourdin	1	Page 25
Infos	Wanted, suite	A. Bonnet/JYB	3	Disq 03
Initiation	Du Basic au Pascal, 1e partie	J. Destelle	5	Page 36
Initiation	Du Basic au Pascal, 2e partie	J. Destelle	6	Page 44
Initiation	Introduction à la programmation du GS	J. Destelle	5	Page 34
Initiation	Introduction aux ressources GS	J. Destelle	6	Page 42
Initiation	La gestion de la mémoire du GS	J. Destelle	5	Page 41
Initiation	La gestion des événements sur GS	J. Destelle	6	Page 39
Initiation	Le GS pour débutants, 1e partie	E. Weyland	4	Page 24
Initiation	Le GS pour débutants, 2e partie	E. Weyland	6	Page 28
Initiation	MiniDesktop	J. Destelle	6	Disq 06A
Logiciel	ResDoctor	J. Destelle	6	Page 26
News	Exotica	F. Hermellin	2	Page 35
News	Exotica	F. Hermellin	4	Page 14
News	Exotica	F. Hermellin	5	Page 20
News	Fumées sans feu	J.Y. Bourdin	1	Page 37
News	GS News	E. Weyland	1	Page 29
News	GS News	E. Weyland	2	Page 42
News	GS News	E. Weyland	3	Page 43
News	GS News	E. Weyland	4	Page 36
News	GS News	E. Weyland	5	Page 45
News	Made in France	B. Fournier	2	Page 34
News	Made in France	B. Fournier	3	Page 27
News	Made in France	B. Fournier	4	Page 33
News	Made in France	B. Fournier	6	Page 24
News	Octobre 1990: Apple relance l'Apple II !	J.Y. Bourdin	2	Page 17

Genre	Titre	Auteur	N°	Situation
Pascal	De TML à Orca-Pascal	M. Racine	4	Page 47
Pascal	Ecrire des Inits en Pascal	B. Fournier	4	Page 13
Pascal	La programmation de Sélect	J. Destelle	5	Page 25
Pascal	Pascal et assembleur	B. Fournier	5	Page 16
Patch	Appleworks Mousetext	D. Skutnik	3	Disq 03
Patch	Patch pour Graphic Exchange	J.Y. Bourdin	1	Disq 01
Patch	TransWarp DisableDataCache	J.Y. Bourdin	3	Disq 03
Poisson	L'égalité Mac-GS, enfin !	Rédaction	4	Page 28
Présentation	Animation N°2	Second Sight	2	Disq 02
Présentation	Animation N°3	Second Sight	3	Disq 03
Présentation	Animation N°4	Second Sight	4	Disq 04
Présentation	Animation N°5	Second Sight	5	Disq 05
Présentation	Animation N°6	Second Sight	6	Disq 06A
Présentation	Quand ToolBox-Mag craque	J.Y. Bourdin	2	Page 50
Présentation	Sommaire de la disquette ToolBox-Mag 1	Rédaction	1	Page 02
Présentation	Sommaire de la disquette ToolBox-Mag 2	Rédaction	2	Page 02
Présentation	Sommaire de la disquette ToolBox-Mag 3	Rédaction	3	Page 02
Présentation	Sommaire de la disquette ToolBox-Mag 4	Rédaction	4	Page 02
Présentation	Sommaire de la disquette ToolBox-Mag 5	Rédaction	5	Page 02
Présentation	Sommaire des disquettes ToolBox-Mag 6	Rédaction	6	Page 02
Programme	Border Scroll	D. Delay	4	Page 22
Programme	Fun Beyond GS, un jeu en six versions	O. Goguel	4	Page 34
Programme	GE 80	R. Mange	4	Page 40
Programme	GetVersion	J.Y. Bourdin	4	Disq 04
Programme	GS Puzzle: un puzzle avec les images GS	B. Boissière	2	Page 37
Programme	Joli.Bureau	P. Desnoues	4	Page 05
Programme	Lynx	P. Desnoues	5	Page 31
Programme	Mac Eraser	O. Goguel	3	Page 28
Programme	Mac Eraser, le source	O. Goguel	4	Page 16
Programme	MasterMind GS, un jeu en TML Pascal	R. Barbieux	3	Page 29
Programme	MidiSynth Player	J.P. Charpentier	4	Page 30
Programme	Mjuke et TML Pascal	J. Destelle	2	Disq 02
Programme	Mjuke, Music Studio Juke-Box, un outil GS	S. Hadinger	1	Page 20
Programme	MuSeq, vers un séquenceur	J.P. Charpentier	6	Page 36
Programme	NDA Toolbox	P. Desnoues	1	Disq 01
Programme	PicMaster GS: convertissez les images GS	B. Fournier	2	Page 26